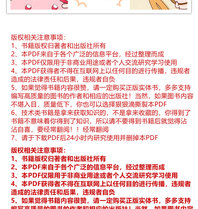
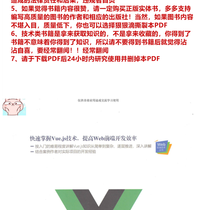
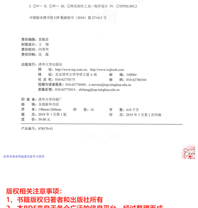






作者简介

华阳锦<http://www.cnblogs.com/yuanyuan/>，软件行业老兵，mailto:yuanyuan@126.com，2001-2005 年在杭州电子科技学院计算机学院获得软件专业学士学位，毕业后在北京工作，曾经工作人员涉及：私企、中型国有企业以及一家上市公司，目前供职于北京时间在北京。

- 2006 年在新人中从事 C++ 开发工作；
- 2007-2008 年在新人中从事 C# 开发工作；
- 2009 年在新人中从事 C# 开发工作；
- 2010-2012 年在新人中从事 C# 开发工作；
- 2013-2014 年在新人中从事 C# 开发工作；
- 2014-2015 年在新人中从事 C# 开发工作；2015 年至今在新人从事 C#/Python 开发工作。

从事软件行业十多年，其间深刻体会到技术变化之快。

- 在系统层次下 Java、Ruby on Rails、Python、分布式体（DevOps）；
- 浏览器层次下 Vanjs、jQuery、CSS；
- 移动系统层次下 Android、Hand Native；

技术变化之快的同时，感觉自己的知识储备一直很匮乏，所以开始学习新技术，比如说 Ruby, Rails, Git，自动化测试，Vim 和软件架构设计等等。

前 言

本书是根据笔者在对适合初学者的实际项目经验编写的。

笔者从 2006 年经营自己的软件公司，到 2010 年关闭，做了近三十个项目，这些项目中，有的采用的 Webpack 开发模式，大部分项目都是采用手动编写 Vue.js 的模式。

在使用 Vue.js 之前，笔者考察过 Angular（包括 1.x 及 4.x）Knot, Meteor, 这几个框架都是功能非常强大，甚至在其中一些框架的功能，概念复杂，把简单的事情变复杂了（如 Angular）。

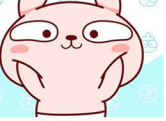
要么就是偏向内容不足，比如当时内容不足一年（如 Knot, Meteor），而 Vuejs 在当时就已经非常成熟了。

Vue.js 的作者尤雨溪在 2016 年 4 月，我国第一次 C3P0 的，明显不是一个 Webpack 开发的前端框架，一样的构建以上手都很好，认真学习的话，一个月就可以达到熟练水平（熟练的开发者），对二个月就可以达到精通水平。

这么简单的一个框架，真的值得学习吗？

这么简单的一个框架，真的值得学习吗？

这么简单的一个框架，真的值得学习吗？



目 录

第 1 章

Vue.js概述

1.1 单页应用的出现

随着移动电话的普及和微信的流行，很多的 Wap（H5）应用也随之出现了，如微店、微站各个 App 中包含的 H5 页面。

手机的硬件特点有：

- 硬件设备差。同主频的手机 CPU 性能往往是台式机的十分之一（手机的供电与台式机设备相差很远）。
- 网络速度慢。4G 网络在很多时候下载速度只有几百 KB，打开一个微信中的网页可能也要很久。

因此，使用传统的 Webpack 技术开发的网页，在手机端的表现往往特别差。传统技术的特点是：

- 单击某个链接/按钮，或者提交表单后，Webpack 页面整体刷新。
- js/css 的请求往往很多，过百是很常见的事情。

每次页面整体刷新，都要导致浏览器重新加载对应的内容，特别“卡顿”。另外，加载的内容也很多。很多传统页面的 css/js 多达上百个，每次打开页面都需要发送上百次请求。如果页面中包含 websocket 等内容，打开速度就会更慢。

苹果的机器表现还好，iOS 设备打开 Web 页面速度很快，Android 设备则大部分都很慢。这个是由手机设备操作系统、软件及智能硬件决定的。

单页应用（Single Page App，SPA）体现出了其强大的优势。

- 页面是局部刷新的，响应速度快，不需要每次加载所有的 js/css。
- 前后端分离，前端（手机端）不受（服务器端的）开发语言的限制。

越来越多的 App 采用 SPA 的架构。如果你的项目要用在 H5 上，那么一定要使用单页应用框架，如 Angular、React、Vue.js 都是很好的框架。

我们在公司实际项目中，都使用 Vue.js，效果非常好。开发速度快，维护效率高。

因为本文与官方文档不同，是根据实际项目经验，以培养新人的角度来写的，所以会有以下特点。

- 很少使用的技术略过。
- 只讲解常见的知识。
- 在章节上按照入门的难易度程度从简单到复杂。

1.2 为什么要使用 Vue.js

在本章中，我们会从多个角度思考这个问题。

1.2.1 单页应用

Web 的应用分两类：传统 Web 页面应用和单页应用（Single Page App）。

1. 传统 Web 页面

传统 Web 页面就是打开浏览器，整个页面都会打开的应用。例如，笔者的个人网站 http://siwei.me 就是一个典型的“传统 Web 应用”，每次单击其中任意一个链接，都会引起页面的整个刷新，如图 1-1 所示。

图 1-1 个人网站

从图 1-1 中可以看出，传统的页面每次打开，都要把页面中的.js、.css、图片文件、html 文件等资源加载一遍。在图 1-1 的左下角可以看到，本次共加载了 10 个请求（4 个 css，2 个 js，3 个 png 图片及 1 个 html 文件），耗时 0.837s。这个在 PC 端可以，但是在手机端就会特别慢，特别是在安卓手机上。

传统页面的特点就是下面任何一个操作，都会引起浏览器对于整个页面的刷新：

- 单击链接。
- 提交表单。
- 触发 location.href='...'这样的 js 代码。

我们来看一个传统 Web 页面的例子。

```
<html>
<head>
   <script src="my.js"></script>
   <style src="my.css"></style>
</head>
<body>
   <img src='my.jpg' />
   <p> 你好！ 传统 Web 页面！ </p>
</body>
</html>
```

每个浏览器都会从第一行解析到最后一行，然后继续加载 my.js、my.css、my.jpg 这三个外部资源。

其实很好理解，这个就是大家最初想象的 Web 页面的打开方式。

2. 单页应用（Single Page App）

单页应用，确切的诞生时间不详，可以肯定的是这个概念 2003 年就在论坛上被人讨论了。在 2002 年 4 月，诞生了一个网站：http://slashdotslash.com，就使用了这种思想。

单页应用的精髓是点/单击任何链接，都不会引起页面的整体刷新，只会通过 JavaScript 替换页面的局部内容。

3. Ajax 和 XML

说到这里，就不得不提到另一个概念：Ajax（Asynchronous JavaScript），中文可以称之为“js 的异步请求”，国内统一称为 Ajax。

Ajax 的概念是每次打开新的网页时，不要让页面整体刷新，而是由 js 发起一个“HTTP 异步请求”，这个“异步请求”的特点就是不让当前的网页“卡”死。

用户可以一边上下滚动页面，播放视频一边等待这个请求返回数据。结果被正常返回

后，由 js 控制刷新页面的局部内容。

这样做的好处是：

（1）大大节省了页面的整体加载时间。 各种.js、.css 等资源文件加载一次就够了。

（2）节省了带宽。

（3）同时减轻了客户端和服务端的负担。

在智能手机和 App 应用（特别是微信）流行起来之后，大量的网页都需要在手机端打开，Ajax 的优势就体现的淋漓尽致。

虽然 Ajax 的名称本意是“异步 js 与 XML”，但是现在在服务器端返回的数据中几乎都使用 JSON，而抛弃了 XML。

在 2005 年，国内的程序员论坛开始提及 Web 2.0，其中 Ajax 技术被人重视。到了 2006 年初，可以说 Ajax 是前端程序员的加薪利器。市面上的所有招聘“前端 Web 程序员”的职位描述中都认为 Ajax 是重要的加分项。

可惜当时 jQuery 在国内不是很普及，Prototype 也没有流行起来。笔者与北京软件圈子里的各大公司的同行们交流时，发现大家用的都是“原生的 JavaScript Ajax”，这种不借助任何第三方框架的代码写起来非常臃肿、累人，而且考虑到浏览器的兼容问题，开发起来也很让人头疼。

例如，当时的代码往往是这样的：

```
var xmlHttp = false;
/*@cc_on @*/
/*@if (@_jscript_version >= 5)
try {
    xmlHttp = new ActiveXObject("Msxml2.XMLHTTP");
} catch (e) {
    try {
        xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
    } catch (e2) {
        xmlHttp = false;
    }
}
@end @*/
if (!xmlHttp && typeof XMLHttpRequest != 'undefined') {
    xmlHttp = new XMLHttpRequest();
}
```

上面的代码仅仅是为了兼容各种浏览器。实际上，后面还有几十行的冗余代码，之后才是正常的业务逻辑代码。

到 2008 年，国内开始流行 Prototype、jQuery 之后，发起一个 Ajax 请求的代码精简成几行：

```
jQuery.get('http://some_url?para=1', function(data){
    // 正常代码
})
```

那时候开始，Ajax 在国内变得越来越普及。

4. Angular

第一个单页应用（SPA）的知名框架应该是 Angular，由 Google 在 2010 年 10 月推出。当时的 Gmail、Google Map 等应用对于 Ajax 技术运用到了极致。 而 Angular 框架一经推出，立刻引燃了单页应用这个概念。尽管后来各种 SPA 框架层出不穷，在 2015 年之前，Angular 稳坐 SPA 的头把交椅。

5. 当下的 SPA 技术趋势

SPA 框架已经成为了项目开发必不可少的内容，只要有移动端开发，就会面临以下两个选择。

- 做成原生 App。
- 做成 SPA H5。

无论是 iOS 端还是 Android 端，都对 SPA 青睐有加。

（1）打开页面速度特别快。打开传统页面，手机端往往需要几秒，而 SPA 则在 0.x 秒内。

（2）耗费的资源更少。因为每次移动端只请求接口数据和必要的图片资源。

（3）对于点/单击等操作响应更快。对于传统页面，手机端的浏览器在操作时，点击按钮会有 0.1s 的卡顿，而使用 SPA 则不会有卡顿的感觉。

（4）可以保存浏览的历史和状态。不是每一个 Ajax 框架都有这个功能。例如，QQ 邮箱，虽然也是页面的局部刷新， 但是每次打开不同的邮件时， 浏览器的网址不会变化。而在所有的 SPA 框架中，都会有专门处理这个问题的模块，叫做 router（路由）。

例如：

http://mail.my.com/#/mail_from_boss_on_0620 对应老板在 6 月 20 日发来的邮件。

http://mail.my.com/#/mail_from_boss_on_0622 对应老板在 6 月 22 日发来的邮件。

在 2011 和 2012 年，各种 SPA 框架出现了井喷的趋势，包括 Backbone、Ember.js 等上百个不同的框架。近几年，比较流行的框架是 Angular、React 和 Vue.js。

1.2.2　知名的单页应用（SPA）框架对比

在学习 Vue.js 之前，我们要知道为什么学习它。

目前市面上比较知名的单页应用 SPA 框架是 Vue.js、React、Angular，我们依次来了解一下。

1. Angular

Angular 作为 SPA 的老大哥，源于 Google，在过去若干年发挥了非常大的价值。现在的版本是 4.0

它的优点是：

- 业内第一个 SPA 框架。
- 实现了前端的 MVC 解耦。
- 双向绑定。Model 层的数据发生变化会直接影响 View，反之亦然。

缺点也很明显：

- 难学，难用。
- Angular 1.x 的文档很差。2.0 稍微好一些。

Angular 1.x 的文档 directive 被无数人吐槽看不懂。文档不全，没有示例代码。很多东西调试起来也没有专门的工具。另外，想使用第三方组件的话，需要单独为 Angular 做适配。例如，jquery-upload 前端上传文件的组件，非常不好用。

虽然 Angular 的功能很全面，但是由于学习曲线过于陡峭，上手很慢，维护起来也很麻烦。因此，现在在论坛上的口碑也开始下降。官方网站为 https://github.com/angular。

2. React

React 是由 Facebook 推出的 SPA 框架，宣称的特点是 Learn once, write anywhere，很吸引人。

React 的优点是：

- 使用 js 一种语言就可以写前端（H5、App）+后端。
- ReactNative 可以直接运行在手机端，性能很棒，接近于原生 App，并且可以热更新，免去了手机端 App 每次都要重新下载和安装的过程。
- 周边组件很多。

React 的缺点是：

（1）html 代码（这样的标签）需要写在 js 文件中。例如：

```
class HelloMessage extends React.Component {
  render() {
    return (
      <div>
        Hello {this.props.name}
      </div>
    );
  }
}
```


```
ReactDOM.render(
  <HelloMessage name="Taylor" />,
  mountNode
);
```

上面代码的编程方式也叫“多语言混合式编程”。最大的特点是代码难以理解、开发和调试。

（2）把前后端代码写在一起的风格。

```
//前端代码

....


// 后端代码
....
```

所有做过传统 Web 框架的人都觉得奇怪。其他表现尚可，学习难度低于 Angular，高于 Vue.js。官方网站为 https://github.com/facebook/react。

3. Vue.js

Vue.js（读音同 View）是一个 MVVM（Model - View - ModelView）的 SPA 框架。

- View：视图。
- Model：数据。
- ModelView：连接 View 与 Model 的纽带。

Vue.js 一经推出，就获得了各大社区的好评，几乎是一边倒的声音。它的优点是：

（1）简单好学，好用。

- Angular：学习二周到四周。
- React：学习两周。
- Vue.js：三天到一周。

这三个框架做的事都一样。

（2）Angular、React 具备的功能，Vue.js 都具备（React Native 除外）。

Vue.js 在 2014 年 2 月被推出的时候，核心文档就具备了两种语言：中文和英文，这对于母语是汉语的国人来说意义重大，可以非常快的上手。官方网站为 https://github.com/Vuejs/vue。

4. 为什么用 Vue.js，不用 React、Angular

首先，我们在评价一个技术的时候，最简单的办法就是看它有多火，这个体现在 Github 的 stars 数目上。

截止到 2018 年 6 月底，三个项目的关注数分别是：

- Angular：3.8 万。
- React：10.4 万。
- Vue.js：10.5 万。

可以看出，Vue.js 排在第一位。

其次，我们看一下 stars 的增长趋势。

根据统计（http://www.timqian.com/star-history/#facebook/react&angular/ angular&Vue.js/vue），截止到 2018 年 6 月底，Vue.js 的增长趋势一直是最高的，React 居中，Angular 最低，如图 1-2 所示的 github 上三种框架的关注变化曲线。

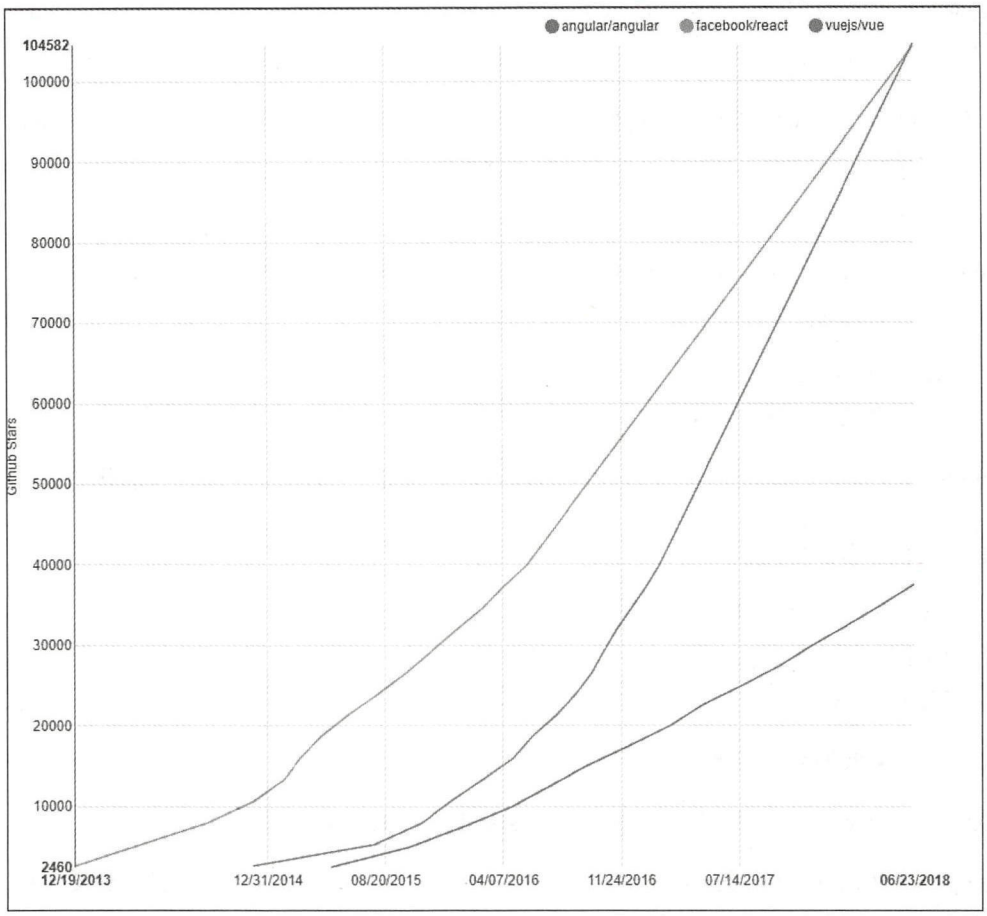


图 1-2　github 上三种框架的关注变化曲线

第三，Vue.js 的作者是中国人，官方文档也是中文的（地址：http://cn.Vue.js.org）。

读者可以参考以下两篇文章。

- https://cn.Vue.js.org/v2/guide/comparison.html;
- https://www.quora.com/How-does-Vue-js-compare-to-React-js。

1.2.3 被腾讯和阿里巴巴所青睐

Vue.js 的思想可以说是对国内互联网巨头产生了较大的冲击。腾讯的微信和阿里巴巴的 Weex 项目的实现方式与 Vue.js 是非常相似的。

可以说，学会了 Vue.js 就基本学会了微信小程序和阿里巴巴的 Weex。这个对于需要不断学习新知识的程序员来说，是非常好的消息。有大公司的支持，这个技术一定是非常有前景的。

1. 微信小程序

微信小程序是微信在 2017 年出现的技术，基于微信。使用 SPA 的开发技术，就可以运行在安装过微信的手机上。表现效果与原生 App 几乎一样。

微信小程序的代码特点、文件组织形式及各种概念，与 Vue.js 是非常相似的。

2. 阿里巴巴 Weex

Weex 致力于使开发者能基于当代先进的 Web 开发技术，使用同一套代码来构建 Android、iOS 和 Web 应用。

Weex 以及支持了对于 Vue.js 2.0 的直接集成。也就是可以在 Weex 中直接写 Vue.js 的代码。Weex 与 React Native 一样，都是使用 Web 开发的相似代码，让程序以 Native App 的形式跑起来。

Weex 以官方文档的形式告诉大家如何使用 Vue.js，链接如下：

```
https://weex.apache.org/cn/guide/use-vue.html 。
```

1.2.4 用到 Vue.js 的项目

- 滴滴出行。
- 饿了么：开源了一个基于 Vue 的 UI 库（https://github.com/ElemeFE/element）。
- 阿里巴巴的 weex（https://github.com/alibaba/weex）。
- GitLab（https://about.gitlab.com/2016/10/20/why-we-chose-vue/）。
- facebook（ https://newsfeed.fb.com/welcome-to-news-feed?lang=en）。
- 新浪微博。

更全列表可见 https://github.com/Vue.js/awesome-vue#projects-using-Vue.js。

第 2 章

原生的Vue.js

所谓的原生 Vue.js，就是独立的 Vue.js 框架，不与 Webpack 等框架结合使用。学习这个比较有必要，因为在后期查看官方文档时，很多概念都是用“原生 Vue.js”的形式说明的，脱离了其他框架，说明起来更加简明一些。

虽然我们在做项目时极少会使用原生 Vue.js，但是对于未来的学习大有好处。

2.1 极速入门

如果只从体验的角度来看，Vue.js 的安装非常简单，只需要引入一个第三方的 js 包即可。

```
<script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
```

下面是一个简单的例子。

```
<html>
<head>
    <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
</head>
<body>
    <div id='app'>

    </div>
    <script>
        var app = new Vue({
            el: '#app',
            data: {
                show_my_text: 'Vue.js is the best one page App!'
            }
        })
```

```
    </script>
</body>
</html>
```

上面的代码非常简单。

（1）在 head 中引入 Vue.js 包。

（2）在<body>中，定义了一个 ，可以认为，所有的页面展示都是在这个 中。每次我们做任何点/单击的时候，整个页面不会刷新，都是 Vue.js 框架操作代码，对其中的内容进行局部刷新。

（3）后面的 var app=new Vue...就是真正的操作代码。

```
// 表示创建了一个 Vue 对象
var app = new Vue({
   // 指定所有的代码操作，都是对于 <div id='app' > 的元素来操作的
    el: '#app',
   // data 表示为 Vue.js 管理的变量赋值。这个变量的名字是 show_my_text
    data: {
        show_my_text: 'Vuejs is the best one page app!'
    }
})
```

（4）使用浏览器打开这个页面后，就可以看到如图 2-1 所示的页面。

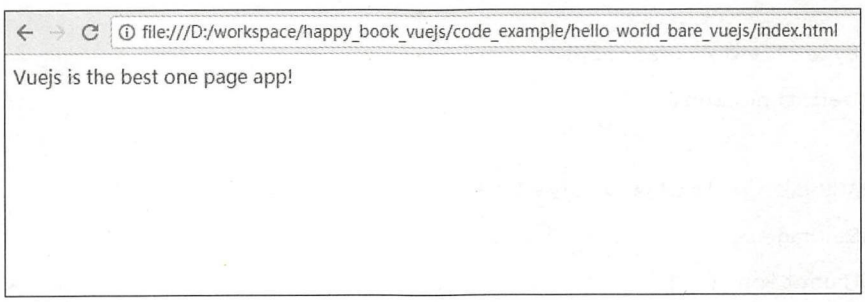


图 2-1　页面效果

这个页面的源代码可以在 code_example/hello_world_bare_Vue.js 中看到并直接运行。

2.2 实际项目

下面我们看一个实际的例子。这个例子就是 SPA 的应用：TODO-list，来自于 Vue.js 的官方。

需求如下：

（1）可以列出代办事项。

（2）可以新增代办事项。

（3）可以把待办事项标记成“已办完”。

该例子的目的是为了让大家对于原生的 Vue.js 有一个直观的认识，里面的技术细节其实有些复杂，使用了基本的 Vue.js 知识、Component（组件）、Watcher（监听器）、Computed Properties（计算得到的属性）、Filter（过滤器）等概念。读者暂时不用深究，在第 4 章 Webpack+Vue.js 实战中会依次讲解到。

读者只需要对实际的原生项目有所了解即可。

2.2.1 运行整个项目

新建文件 index.html，内容如下：

```
<!DOCTYPE html>
<head>
  <meta http-equiv="content-type" content="text/html; charset=UTF-8">
  <link rel="stylesheet" type="text/css" href="https://unpkg.com/todomvc-app-
css@2.0.6/index.css">
  <script type="text/javascript"
src="https://unpkg.com/vue@latest/dist/vue.js"></script>

  <script type="text/javascript">
window.onload=function(){

  var STORAGE_KEY = 'todos-vuejs-2.0'
  var todoStorage = {
    fetch: function () {
      var todos = JSON.parse(localStorage.getItem(STORAGE_KEY) || '[]')
      todos.forEach(function (todo, index) {
        todo.id = index
      })
      todoStorage.uid = todos.length
      return todos
    },
    save: function (todos) {
      localStorage.setItem(STORAGE_KEY, JSON.stringify(todos))
```

```
  }
}

// visibility filters
var filters = {
  all: function (todos) {
    return todos
  },
  active: function (todos) {
    return todos.filter(function (todo) {
      return !todo.completed
    })
  },
  completed: function (todos) {
    return todos.filter(function (todo) {
      return todo.completed
    })
  }
}

// app Vue instance
var app = new Vue({
  // app initial state
  data: {
    todos: todoStorage.fetch(),
    newTodo: '',
    editedTodo: null,
    visibility: 'all'
  },

  // watch todos change for localStorage persistence
  watch: {
    todos: {
      handler: function (todos) {
        todoStorage.save(todos)
      },
```

```
    deep: true
  }
},

computed: {
  filteredTodos: function () {
    return filters[this.visibility](this.todos)
  },
  remaining: function () {
    return filters.active(this.todos).length
  },
  allDone: {
    get: function () {
      return this.remaining === 0
    },
    set: function (value) {
      this.todos.forEach(function (todo) {
        todo.completed = value
      })
    }
  }
},

filters: {
  pluralize: function (n) {
    return n === 1 ? 'item' : 'items'
  }
},

// methods that implement data logic.
// note there's no DOM manipulation here at all.
methods: {
  addTodo: function () {
    var value = this.newTodo && this.newTodo.trim()
    if (!value) {
      return
```

```
    }
    this.todos.push({
      id: todoStorage.uid++,
      title: value,
      completed: false
    })
    this.newTodo = ''
  },

  removeTodo: function (todo) {
    this.todos.splice(this.todos.indexOf(todo), 1)
  },

  editTodo: function (todo) {
    this.beforeEditCache = todo.title
    this.editedTodo = todo
  },

  doneEdit: function (todo) {
    if (!this.editedTodo) {
      return
    }
    this.editedTodo = null
    todo.title = todo.title.trim()
    if (!todo.title) {
      this.removeTodo(todo)
    }
  },

  cancelEdit: function (todo) {
    this.editedTodo = null
    todo.title = this.beforeEditCache
  },

  removeCompleted: function () {
    this.todos = filters.active(this.todos)
```

```
    }
  },

  directives: {
    'todo-focus': function (el, binding) {
      if (binding.value) {
        el.focus()
      }
    }
  }
})

// handle routing
function onHashChange () {
  var visibility = window.location.hash.replace(/#\/?/, '')
  if (filters[visibility]) {
    app.visibility = visibility
  } else {
    window.location.hash = ''
    app.visibility = 'all'
  }
}

window.addEventListener('hashchange', onHashChange)
onHashChange()

app.$mount('.todoapp')
}
</script>

</head>
<body>
  <section class="todoapp">
  <header class="header">
    <h1>todos</h1>
    <input class="new-todo"
```


```
      autofocus autocomplete="off"
      placeholder="What needs to be done?"
      v-model="newTodo"
      @keyup.enter="addTodo">
    </header>
    <section class="main" v-show="todos.length" v-cloak>
      <input class="toggle-all" type="checkbox" v-model="allDone">
      <ul class="todo-list">
        <li v-for="todo in filteredTodos"
          class="todo"
          :key="todo.id"
          :class="{ completed: todo.completed, editing: todo == editedTodo }">
          <div class="view">
            <input class="toggle" type="checkbox" v-model="todo.completed">
            <label @dblclick="editTodo(todo)">{{ todo.title }}</label>
            <button class="destroy" @click="removeTodo(todo)"></button>
          </div>
          <input class="edit" type="text"
            v-model="todo.title"
            v-todo-focus="todo == editedTodo"
            @blur="doneEdit(todo)"
            @keyup.enter="doneEdit(todo)"
            @keyup.esc="cancelEdit(todo)">
        </li>
      </ul>
    </section>
    <footer class="footer" v-show="todos.length" v-cloak>
      <span class="todo-count">
        <strong>{{ remaining }}</strong> {{ remaining | pluralize }} left
      </span>
      <ul class="filters">
        <li><a href="#/all" :class="{ selected: visibility ==
'all' }">All</a></li>
        <li><a href="#/active" :class="{ selected: visibility ==
'active' }">Active</a></li>
        <li><a href="#/completed" :class="{ selected: visibility ==
```

```
'completed' }">Completed</a></li>
    </ul>
    <button class="clear-completed" @click="removeCompleted" v-
show="todos.length > remaining">
      Clear completed
    </button>
  </footer>
</section>

</body>
</html>
```

将该文件保存后，使用浏览器直接打开，就可以看到效果。原生 Vue.js 的 TODO-list 项目界面如图 2-2 所示。

图 2-2　原生 Vue.js 的 TODO-list 项目界面

2.2.2　HTML 代码的<head>部分

注释如下所示。

```
<!DOCTYPE html>
<head>
  <!--设置页面的 UTF-8 编码格式-->
  <meta http-equiv="content-type" content="text/html; charset=UTF-8">
  <!-- 引入了对应的 CSS 文件-->
  <link rel="stylesheet" type="text/css" href="https://unpkg.com/todomvc-app-
```

```
css@2.0.6/index.css">
  <!—引入了对应的 vue.js 外部文件 -->
  <script type="text/javascript"
src="https://unpkg.com/vue@latest/dist/vue.js"></script>
  <script type="text/javascript">
// 下面的 window.onload 函数暂时省略。 会在 2.2.3 js 代码部分讲解到。
window.onload=function(){..}
}
</script>
</head>
<body>
  <!-- 这里的代码暂时省略。会在 2.2.2 HTML 代码的 body 部分中讲解到 -->
</body>
</html>
```

2.2.3　HTML 代码的<body>部分

注释如下所示。

```
<body>
  <section class="todoapp">
  <!-- 这里定义了 header，包括了文字的提示和输入框。 -->
  <header class="header">
    <h1>todos</h1>
    <input class="new-todo"
      autofocus autocomplete="off"
      placeholder="What needs to be done?"
      v-model="newTodo"
      @keyup.enter="addTodo">
  </header>
  <!-- 这里的代码把用户已经输入的待办事项做循环展示。  -->
  <section class="main" v-show="todos.length" v-cloak>
    <!--  全选框 -->
<input class="toggle-all" type="checkbox" v-model="allDone">
    <!--  v-for 用来循环 -->
    <ul class="todo-list">
      <li v-for="todo in filteredTodos"
```

```
        class="todo"
        :key="todo.id"
        :class="{ completed: todo.completed, editing: todo == editedTodo }">
        <div class="view">
          <input class="toggle" type="checkbox" v-model="todo.completed">
          <label @dblclick="editTodo(todo)">{{ todo.title }}</label>
          <button class="destroy" @click="removeTodo(todo)"></button>
        </div>
        <input class="edit" type="text"
          v-model="todo.title"
          v-todo-focus="todo == editedTodo"
          @blur="doneEdit(todo)"
          @keyup.enter="doneEdit(todo)"
          @keyup.esc="cancelEdit(todo)">
      </li>
    </ul>
  </section>
  <!-- 底部的状态，点击后可以进行过滤。 -->
  <footer class="footer" v-show="todos.length" v-cloak>
    <span class="todo-count">
      <strong>{{ remaining }}</strong> {{ remaining | pluralize }} left
    </span>
    <ul class="filters">
      <li><a href="#/all" :class="{ selected: visibility ==
'all' }">All</a></li>
      <li><a href="#/active" :class="{ selected: visibility ==
'active' }">Active</a></li>
      <li><a href="#/completed" :class="{ selected: visibility ==
'completed' }">Completed</a></li>
    </ul>
    <button class="clear-completed" @click="removeCompleted" v-
show="todos.length > remaining">
      Clear completed
    </button>
  </footer>
</section>
```


```
</body>
```

2.2.4　js 代码部分

注释如下所示。

```
  <script type="text/javascript">
window.onload=function(){

  // 下面的代码，表示使用了浏览器的 localStorage 进行存储。 同时定义了 fetch() 和 save()
  // 用来方便操作。
  var STORAGE_KEY = 'todos-vuejs-2.0'
  var todoStorage = {
    fetch: function () {
      var todos = JSON.parse(localStorage.getItem(STORAGE_KEY) || '[]')
      todos.forEach(function (todo, index) {
        todo.id = index
      })
      todoStorage.uid = todos.length
      return todos
    },
    save: function (todos) {
      localStorage.setItem(STORAGE_KEY, JSON.stringify(todos))
    }
  }

  // 定义过滤器，可以分别对 ative、completed、和 all 进行过滤。
  var filters = {
    all: function (todos) {
      return todos
    },
    active: function (todos) {
      return todos.filter(function (todo) {
        return !todo.completed
      })
    },
    completed: function (todos) {
```


```
    return todos.filter(function (todo) {
      return todo.completed
    })
  }
}

// 在这里定义 Vue 的实例。
var app = new Vue({
  data: {
    todos: todoStorage.fetch(),
    newTodo: '',
    editedTodo: null,
    visibility: 'all'
  },

  // 定义 watcher
  watch: {
    todos: {
      handler: function (todos) {
        todoStorage.save(todos)
      },
      deep: true
    }
  },
  // 定义 computed properties，可以认为是用来过滤的。
  computed: {
    filteredTodos: function () {
      return filters[this.visibility](this.todos)
    },
    remaining: function () {
      return filters.active(this.todos).length
    },
    allDone: {
      get: function () {
        return this.remaining === 0
      },
```

```
      set: function (value) {
        this.todos.forEach(function (todo) {
          todo.completed = value
        })
      }
    }
  },
  // 定义过滤器
  filters: {
    pluralize: function (n) {
      return n === 1 ? 'item' : 'items'
    }
  },

// 核心方法，实现对于待办事项的增加、删除和修改。
// 注意，这里没有任何直接操作 HTML DOM 的代码。
  methods: {
    addTodo: function () {
      var value = this.newTodo && this.newTodo.trim()
      if (!value) {
        return
      }
      this.todos.push({
        id: todoStorage.uid++,
        title: value,
        completed: false
      })
      this.newTodo = ''
    },
    // 删除待办事项
    removeTodo: function (todo) {
      this.todos.splice(this.todos.indexOf(todo), 1)
    },
    // 编辑待办事项
    editTodo: function (todo) {
      this.beforeEditCache = todo.title
```

```
      this.editedTodo = todo
    },
    // 把待办事项标记为：已完成
    doneEdit: function (todo) {
      if (!this.editedTodo) {
        return
      }
      this.editedTodo = null
      todo.title = todo.title.trim()
      if (!todo.title) {
        this.removeTodo(todo)
      }
    },
    // 取消编辑
    cancelEdit: function (todo) {
      this.editedTodo = null
      todo.title = this.beforeEditCache
    },
    // 删除已经完成的待办事项
    removeCompleted: function () {
      this.todos = filters.active(this.todos)
    }
  },
  // 使用了自定义的 Directive.  也就是 HTML 中的<todo-focus>
  directives: {
    'todo-focus': function (el, binding) {
      if (binding.value) {
        el.focus()
      }
    }
  }
})
// 处理路由
function onHashChange () {
  var visibility = window.location.hash.replace(/#\/?/, '')
  if (filters[visibility]) {
```

```
      app.visibility = visibility
    } else {
      window.location.hash = ''
      app.visibility = 'all'
    }
  }
  window.addEventListener('hashchange', onHashChange)
  onHashChange()
  // 最后，使用 Vue.js 渲染整个页面。
  app.$mount('.todoapp')
}
</script>
```

2.2.5　小结

该例子总共 217 行，代码精炼、功能齐备。可以看出，使用 Vue.js 做开发效率非常高。

完整的代码可见：https://cn.vuejs.org/v2/examples/todomvc.html。

但是，一旦项目的需求增加，代码也会越来越膨胀，把 html, js 和 css 代码都写在一个文件中不是好主意，所以需要以一种更好的形式来组织文件，这就是 Webpack 框架下的 Vue.js。

第 3 章

Webpack+Vue.js开发准备

所有的 Vue.js 项目都是在 Webpack 的框架下进行开发的。可以说 vue-cli 直接把 Webpack 做了集成。在开发时，一边享受着飞一般的开发速度，一边体验着 Webpack 带来的便利。

本章将介绍如何使用 Webpack+Vue.js 进行开发的基本知识。

3.1 学习过程

在学习任何一种框架的时候，都是按照循序渐进的顺序来的。

（1）安装。
（2）新建一个页面。
（3）做一些简单变量的渲染。
（4）实现页面的跳转（路由）。
（5）实现页面间的参数传递（路由）。
（6）实现真实的 http 请求（访问接口）。
（7）提交表单。
（8）使用一些技巧让代码层次化（组件）。

按照笔者之前在惠普、联通、移动等公司的讲课经验，只需要一天的时间就可以把本章内容学会，并且上手开发 Vue.js 项目。

3.1.1 可以跳过的章节

对于有一定 Node 基础的读者，可以跳过“NVM 的安装”。对于使用 Linux/Mac 的读者，可以跳过“Git Bash 的安装”。

3.1.2 简写说明

由于本章是 Webpack + Vue.js 下的实战开发，所以统一使用 Vue.js 来代替冗长的 Webpack + Vue.js。

例如，在 Vue.js 中创建页面需要以下两步。

（1）新建路由。

（2）新建 vue 页面。

3.1.3　本书例子文件下载

本章中的所有例子，由于都需要与 Webpack 相结合，因此笔者将其单独做成一个项目：https://github.com/sg552/vue_js_lesson_demo，读者可以下载后直接运行。

```
$ git clone https://github.com/sg552/vue_js_lesson_demo.git
$ npm install -v
$ npm run dev
```

可以在 http://localhost:8080/#/中看到效果，如图 3-1 所示。

图 3-1　页面效果

3.2　NVM、NPM 与 Node

NVM（Node Version Manager）是非常好用的 Node 版本管理器。这个技术出现的原因，是由于不同的项目 node 版本也不同，有的是 5.0.1，有的是 6.3.2。如果 node 版本不对，运行某个应用时，很可能就会遇到各种莫名其妙的问题。

因此，我们需要在同一台机器上同时安装多个版本的 Node。NVM 应用而生，很好地帮我们解决了这个问题。

Linux/Mac 下的 NVM 官方网址：https://github.com/creationix/nvm。

Windows 下的 NVM 官方网址：https://github.com/coreybutler/nvm-windows。

NPM（Node Package Manager）只要安装了 node，就会捆绑安装该命令。它的作用与

Ruby 中的 bundler 及 Java 中的 maven 相同，都是对第三方依赖进行管理的。

3.2.1 Windows 下的安装

步骤 01 使用浏览器打开 https://github.com/coreybutler/nvm-windows/releases，如图 3-2 所示。

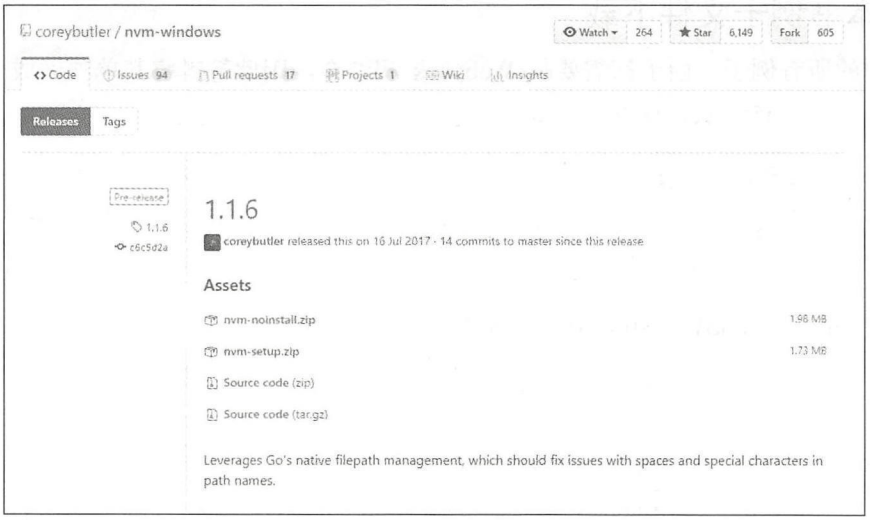


图 3-2 打开下载网址

步骤 02 单击最新的 release 版本进行下载“1.1.6 中的 nvm-setup.zip”文件。

步骤 03 解压下载文件，双击其中的 nvm-setup.exe 文件，就可以开始安装了，如图 3-3 所示。

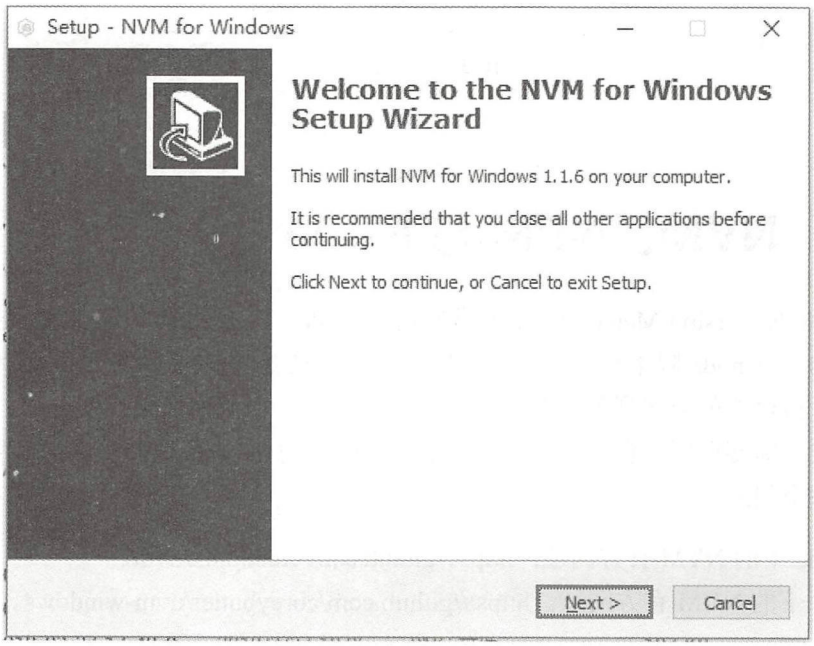


图 3-3 开始安装

步骤 04 单击 Next 按钮，在打开的对话框中选中 I accept the agreement 单选按钮，如图 3-4 所示。

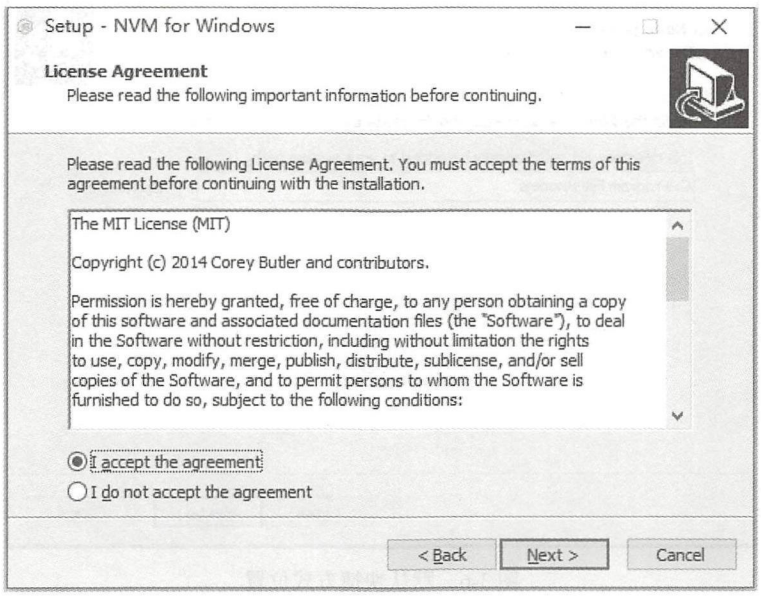


图 3-4　选择接受

步骤 05 继续单击 Next 按钮，在打开的对话框中选择安装路径。这里选择安装到 D:\nvm，如图 3-5 所示。

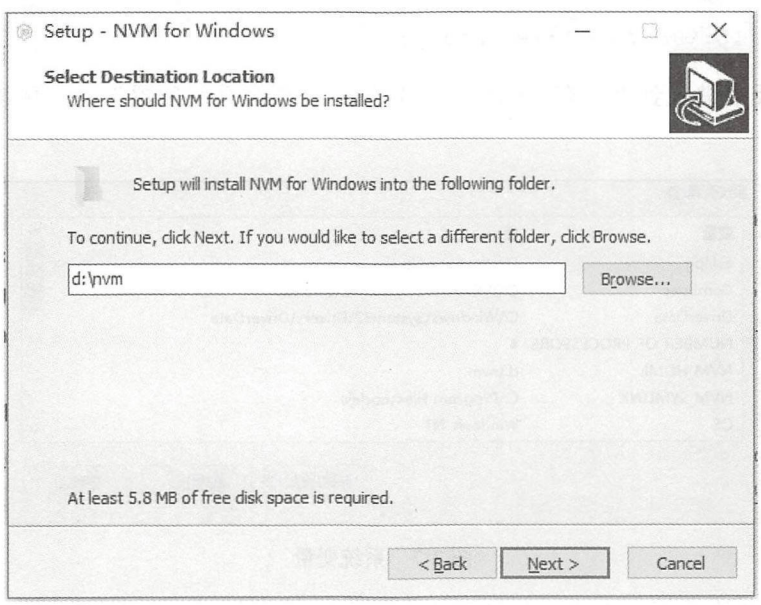


图 3-5　选择安装路径

步骤 06 继续单击 Next 按钮，在打开的对话框中询问把 nvm 的快捷方式放在哪里（symlink 的作用同快捷方式，允许我们在任意路径下都可以调用 nvm 命令），不用修改，直

接单击 Next 按钮，如图 3-6 所示。

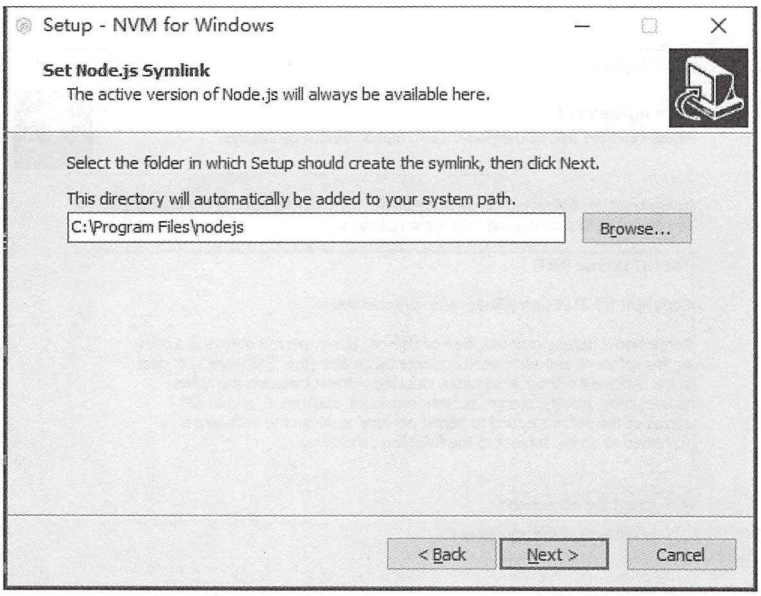


图 3-6　默认快捷方式位置

步骤 07 然后弹出确认安装对话框，继续单击 Next 按钮就可以了。

步骤 08 最后设置环境变量。

```
NVM_HOME          D:\nvm
NVM_SYMLINK       C:\Program Files\nodejs
```

从控制面板中进入到“所有控制面板项目”→“高级系统配置”→“环境变量”，如图 3-7 所示。

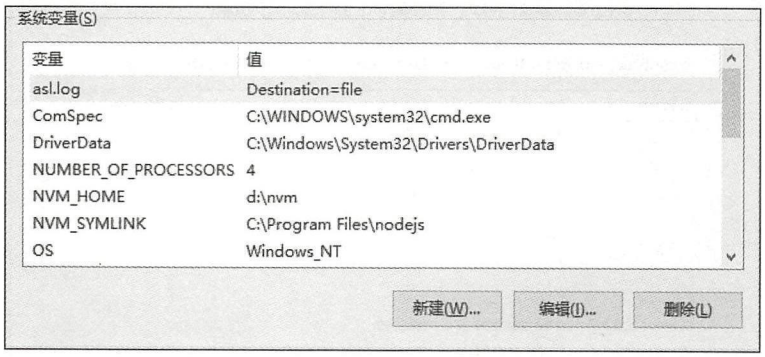


图 3-7　系统变量

对 PATH 的修改则是在原有值的基础上添加%NVM_HOME%、%NVM_SYMLINK%，如图 3-8 所示。

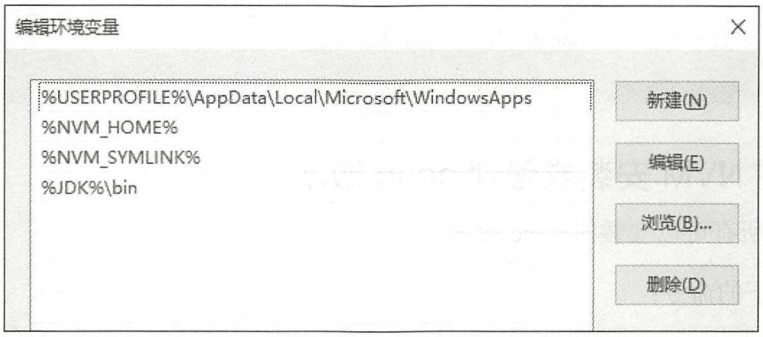


图 3-8　编辑环境变量

3.2.2　Linux、Mac 下的安装

（1）下载 nvm 的源代码，运行下面命令即可。

```
$ git clone https://github.com/creationix/nvm.git ~/.nvm && cd ~/.nvm && git
checkout `git describe --abbrev=0 --tags`
```

（2）Linux、Mac 的用户：为脚本设置启动时加载（对于使用 Windows 的读者，可以直接跳过第二步，到官方网站下载 exe 安装文件就可以了）。

把下面的代码放到~/.bashrc 或~/.bash_profile 或~/.zshrc 中。

```
$ source ~/.nvm/nvm.sh
```

3.2.3　运行

不能使用$ which nvm 验证安装是否成功，因为即使成功了，也不会返回结果。直接在命令行输入以下代码即可。

```
$ nvm
```

如果安装成功，就会看到以下英文。

```
Running version 1.1.6.

Usage:

  nvm arch                     : Show if node is running in 32 or 64 bit mode.
  nvm install <version> [arch] : The version can be a node.js version or
"latest" for the latest stable version....
  nvm list [available]         : List the node.js installations. Type
```

```
"available" at the end to see what can be ...
  nvm on                    : Enable node.js version management.
```

3.2.4 使用 NVM 安装或管理 node 版本

（1）列出所有可以安装的 node 版本。

Windows 下的命令：

```
$ nvm list available
```

Linux/Mac 下的命令：

```
$ nvm list-remote
```

可以看到安装的所有版本。下面是 Windows 中的例子，Linux、Mac 类似。

```
$ nvm list available

|   CURRENT    |    LTS      |  OLD STABLE  | OLD UNSTABLE |
|--------------|-------------|--------------|--------------|
|    10.5.0    |   8.11.3    |   0.12.18    |   0.11.16    |
|    10.4.1    |   8.11.2    |   0.12.17    |   0.11.15    |
|    10.4.0    |   8.11.1    |   0.12.16    |   0.11.14    |
|    10.3.0    |   8.11.0    |   0.12.15    |   0.11.13    |
|    10.2.1    |   8.10.0    |   0.12.14    |   0.11.12    |
|    10.2.0    |   8.9.4     |   0.12.13    |   0.11.11    |
|    10.1.0    |   8.9.3     |   0.12.12    |   0.11.10    |
|    10.0.0    |   8.9.2     |   0.12.11    |    0.11.9    |
|    9.11.2    |   8.9.1     |   0.12.10    |    0.11.8    |

This is a partial list. For a complete list, visit
https://nodejs.org/download/release
```

（2）列出本地安装好的版本。

```
$ nvm list
```

结果形如：

```
$ nvm list
```

```
  * 10.5.0 (Currently using 64-bit executable)
    6.9.1
```

在上面的结果中，表示当前系统安装了两个 node 版本：6.9.1 和 10.5.0。默认的 node 版本是 10.5.0。

（3）安装 node。

选择一个版本号就可以安装了。

```
$ nvm install 10.5.0
```

安装好之后，退出命令行并重新进入即可。

（4）使用 node。

下面的命令是为当前文件夹指定 node 的版本。

```
$ nvm use 10.5.0
```

对于 Linux、Mac，如果希望为系统全局使用某个版本，就可以运行下面的命令。

```
$ nvm alias default 10.5.0
```

在 Linux、Mac 下，还可以将其放到~/.bashrc、~/.bash_profile 中。这样系统每次启动，都会自动指定 node 作为全局的版本。

3.2.5　删除 NVM

对于 Linux、Mac，直接手动删掉对应的配置文件（如果有的话）即可。

- ~/.nvm
- ~/.npm
- ~/.bower

对于 Windows，可直接在控制面板中卸载该软件。

3.2.6　加快 NVM 和 NPM 的下载速度

由于某些原因，在国内连接国外的服务器会比较慢，所以我们使用下面的命令，就可以解决这个问题（默认是使用国外的服务器，现在改成使用国内的镜像服务器）。感谢淘宝提供了这个方法。

对于 NVM，使用 NVM_NODEJS_ORG_MIRROR 这个变量作为前缀。

```
$ NVM_NODEJS_ORG_MIRROR=https://npm.taobao.org/dist nvm install
```

对于 NPM，使用 cnpm 代替 npm 命令。

```
$ npm install -g cnpm --registry=https://registry.npm.taobao.org
```

对于 Linux、Mac 用户，可以直接创建一个"alias"命令。

```
alias cnpm="npm --registry=https://registry.npm.taobao.org \
--cache=$HOME/.npm/.cache/cnpm \
--disturl=https://npm.taobao.org/dist \
--userconfig=$HOME/.cnpmrc"
```

然后通过国内的淘宝服务器安装 node 包。例如：

```
$ cnpm install vue-cli -g
```

3.3 Git 在 Windows 下的使用

在《程序员修炼之道： 从小工到专家》中提到了一个让程序员非常尴尬的局面：老板要看进度，结果程序员拿不出来，只好跟老板撒谎：我的代码被猫吃了。

虽然我们的代码不会被猫吃掉，但是几乎每个程序员都会犯的错误就是：在下班的时候忘记保存，或者突然断电，结果导致写了几个小时的代码就这样没有了。因此，每个程序员必须要对自己的代码做版本控制。

在 2009 年之前，国内的人大部分都用 SVN。从 2010 年开始，越来越多的人开始使用 Git。本节专门为 Windows 程序员准备。因为对于 Linux 和 Mac 用户来说，Git 都是现成的一行命令就能搞定。

3.3.1 为什么要使用 Git Bash

Git Bash 不但提供了 Git，还提供了 bash，一种非常不错的类似于 Linux 的命令行。在 Windows 环境下，命令行模式与 Linux/Mac 是相反的。例如：

- Linux/Mac 下：（使用左斜线作为路径分隔符）

```
$ cd /workspace/happy_book_Vue.js
```

- Windows 下：（使用右斜线作为路径分隔符，并且要分成 C、D 等盘）

```
C:\Users\dashi>d:                          (进入 D 盘)
D:\>cd workspace\happy_book_Vue.js         (进入到对应目录)
```

只要不是做.NET/微信小程序/安卓开发，都应该转移到 Linux 平台上。原因是：代码被编译后，会运行在 Linux+Nginx 的服务器中。最好的办法就是从现在开始就适应 Linux 的环境。另外，命令行在绝大多数情况下比“图形化”的操作界面好用。

3.3.2　安装 git 客户端

在 Windows 下选择 Git Bash。官方网址为 https://gitforwindows.org/。

步骤 01 打开下载页面后就可以看到 Logo，如图 3-9 所示。

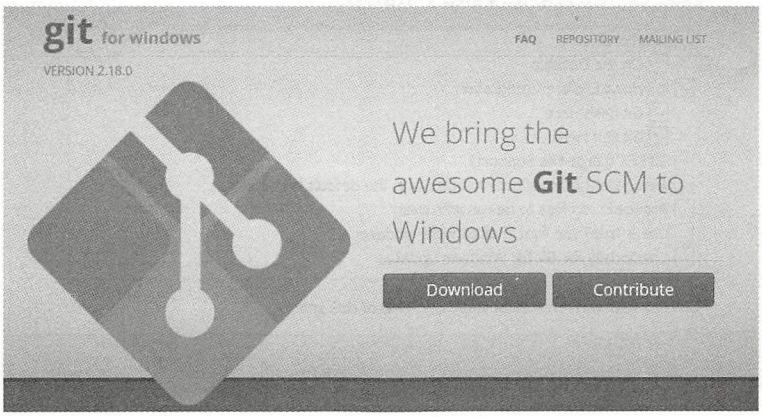


图 3-9　下载页面

步骤 02 选择最新版本，这里下载 2.16.2。

步骤 03 下载并运行，可以看到欢迎对话框，如图 3-10 所示。

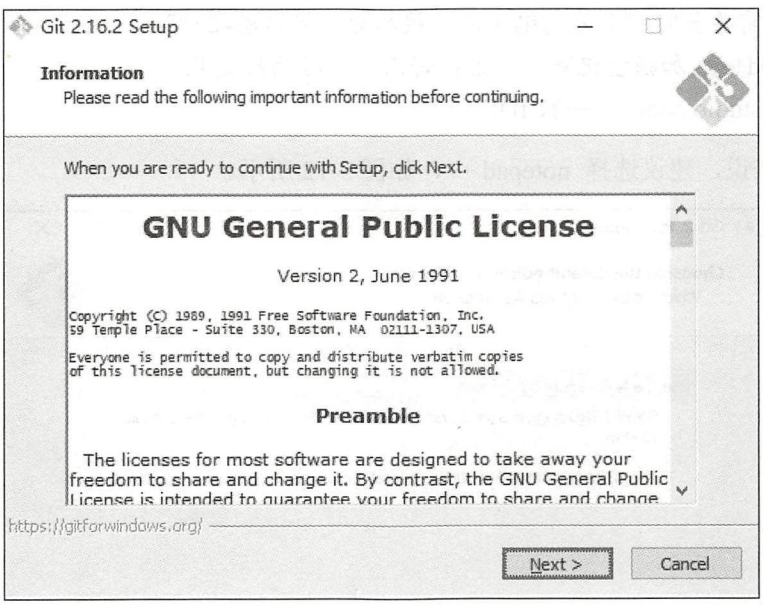


图 3-10　欢迎对话框

步骤 04 单击 Next 按钮，在打开的对话框中看到选择安装的内容，保持默认就好，如图 3-11 所示。

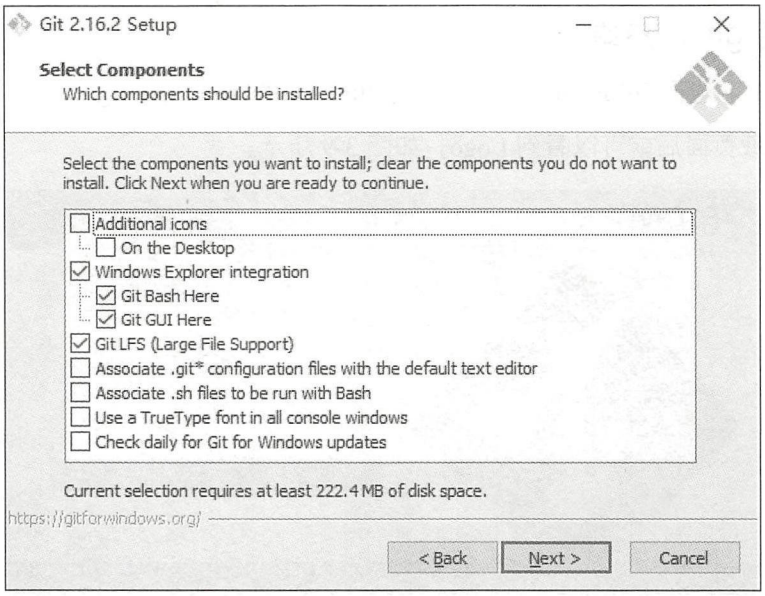


图 3-11　选择安装内容

步骤 05 单击 Next 按钮，在打开的对话框中选择哪个编辑器作为 git 消息编辑器。

- nano：最简单的 Linux 下的编辑器，同 Windows 下的记事本。学习曲线是 0。
- vim：需要长时间学习的编辑器，被称为“编辑器之神”。
- notepad++：加强型记事本，也很好用，学习曲线是 0。
- visual studio code：　一款 IDE。

对于新手来说，建议选择 notepad ++，如图 3-12 所示。

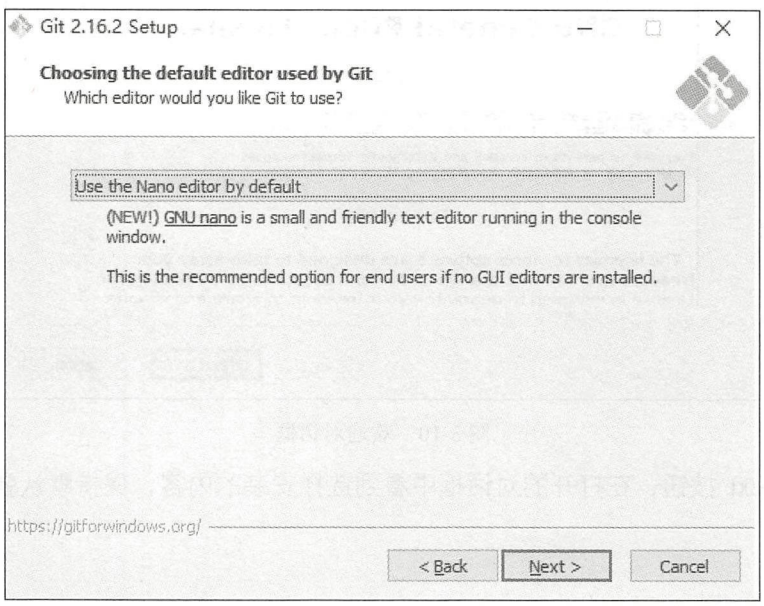


图 3-12　选择编辑器

步骤 06 单击 Next 按钮，询问使用什么风格的命令行。这里建议选择默认的 Use Git from Windows Command Prompt，如图 3-13 所示。

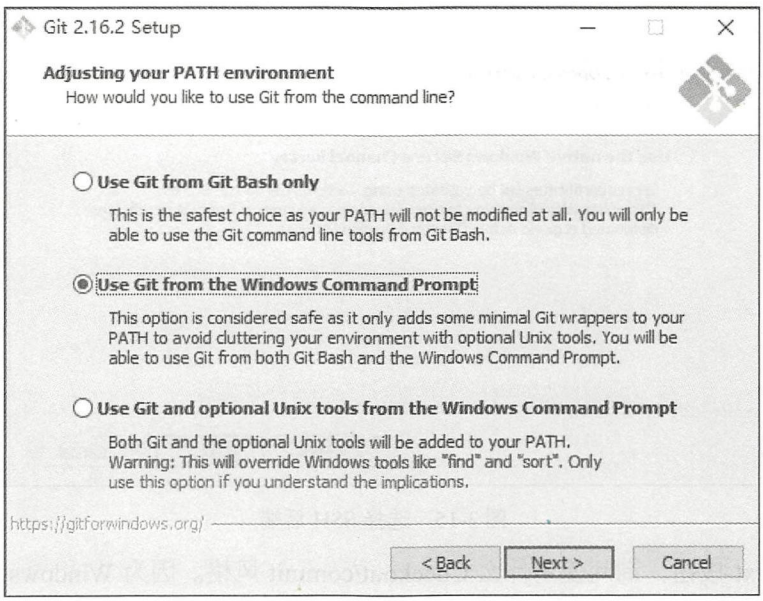


图 3-13　选择默认命令行

步骤 07 单击 Next 按钮，询问使用什么风格的 SSH 连接程序，如图 3-14 所示。

- OpenSSH SSH 的首选，是 Git bash 自带的。
- Plink 第三方用户自己安装的 SSH 连接程序。

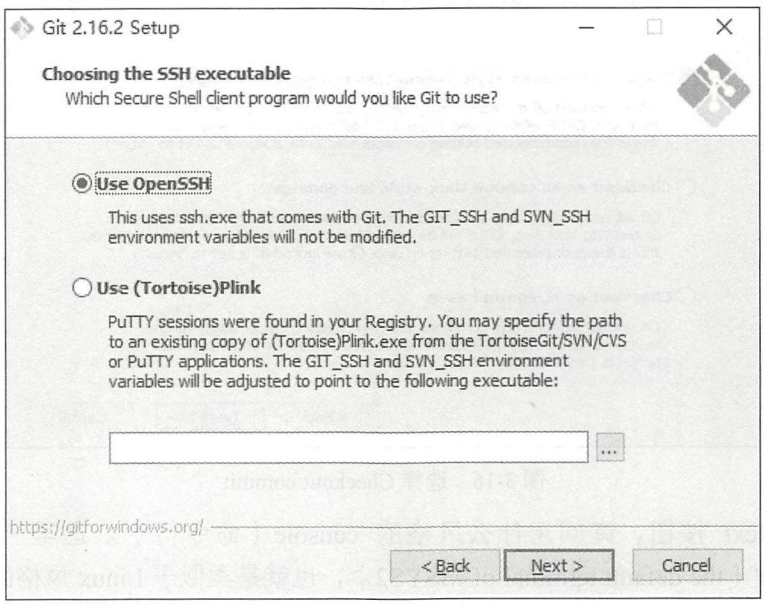


图 3-14　选择 SSH 连接程序

步骤 08 单击 Next 按钮，询问使用什么 SSH 后端，这里选择默认的 OpenSSL，如图 3-15 所示。

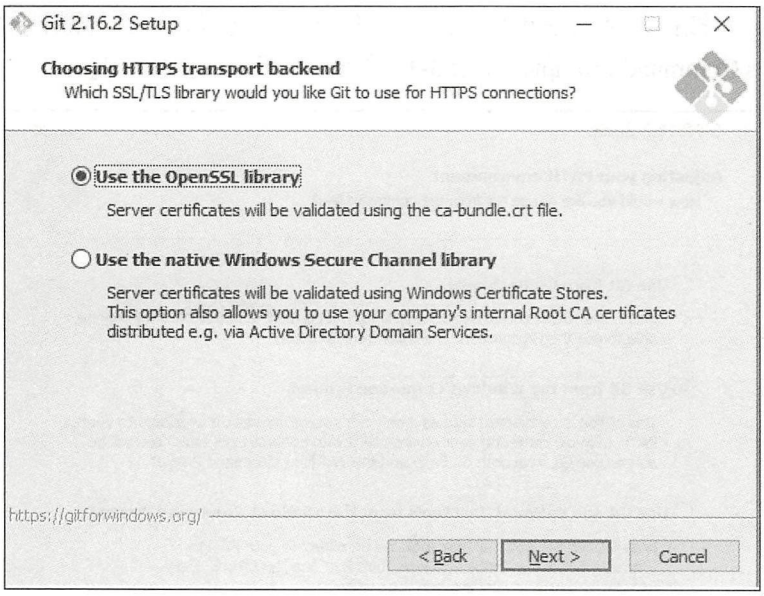


图 3-15　选择 SSH 后端

步骤 09 单击 Next 按钮，询问使用什么 Checkout/commit 风格。因为 Windows 与 linux 对待文件的处理是不同的，如回车在 Windows 下是\r\n，而在 linux 下就是\n，所以这里选择默认的第一项即可（用 Windows 风格 checkout，用 unix 风格 commit），如图 3-16 所示。

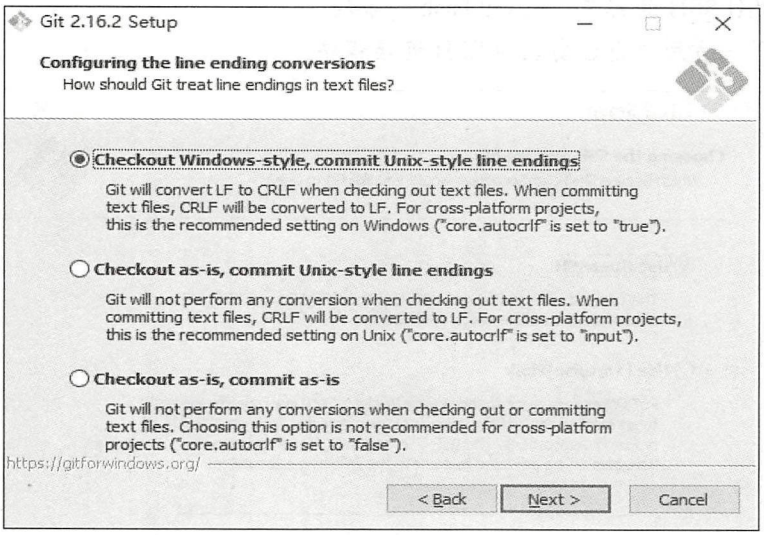


图 3-16　选择 Checkout/commit

步骤 10 单击 Next 按钮，询问用什么风格的 console（命令行）。这里一定要选择 Use MinTTY（the default terminal of MSYS2），也就是类似于 Linux 风格的命令行。可以说，它就是非常著名的 Cygwin，如图 3-17 所示。

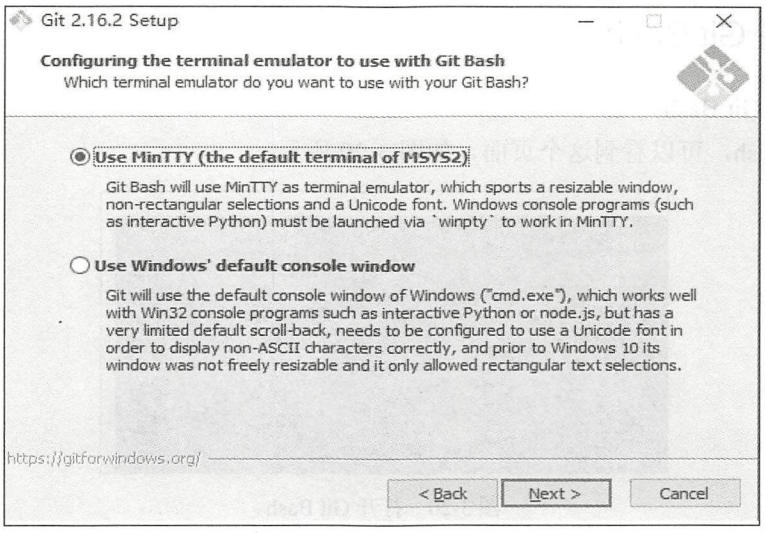


图 3-17　选择 console（命令行）

步骤 11 单击 Next 按钮，询问其他配置项目。直接选择默认即可，如图 3-18 所示。

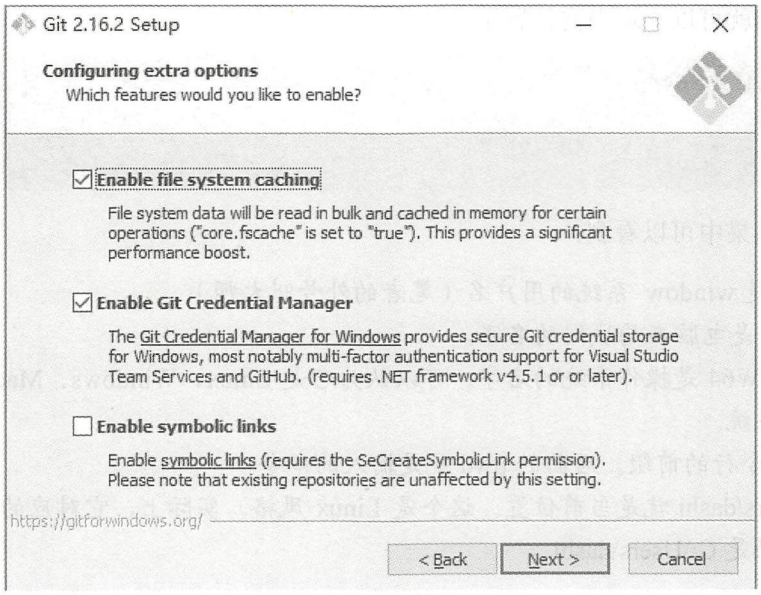


图 3-18　设置其他配置项目

步骤 12 继续单击 Next 按钮，就安装成功了，如图 3-19 所示。

图 3-19　安装成功

3.3.3 使用 Git Bash

（1）打开 Git Bash。

打开 Git Bash，可以看到这个页面，如图 3-20 所示。

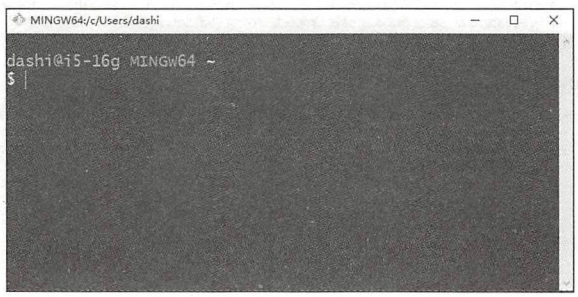


图 3-20　打开 Git Bash

一片空荡荡。估计习惯了鼠标和我的电脑的读者会非常不习惯。不要紧，我们慢慢来。

（2）查看当前路径：pwd。

输入 $ pwd 就可以知道当前位置了。

```
dashi@i5-16g MINGW64 ~
$ pwd
/c/Users/dashi
```

在上面的结果中可以看到：

- dashi 是 window 系统的用户名（笔者的外号叫大师）。
- i5-16g 是电脑在局域网的名字。
- MINGW64 是操作系统的名字。可以认为它是 Linux、Windows、Mac 之外的第 4 种操作系统。
- $是命令行的前缀。后面的 pwd 就是输入的命令。
- /c/Users/dashi 就是当前位置。这个是 Linux 风格。实际上，它对应的 Windows 的标准路径是 C:\Users\dashi

每次打开 Git Bash 的时候，都是默认的“当前用户在 Windows”中的用户文件夹。如果我们在一个窗口中打开这个路径，就可以看到我的用户文件夹了，如图 3-21 所示。

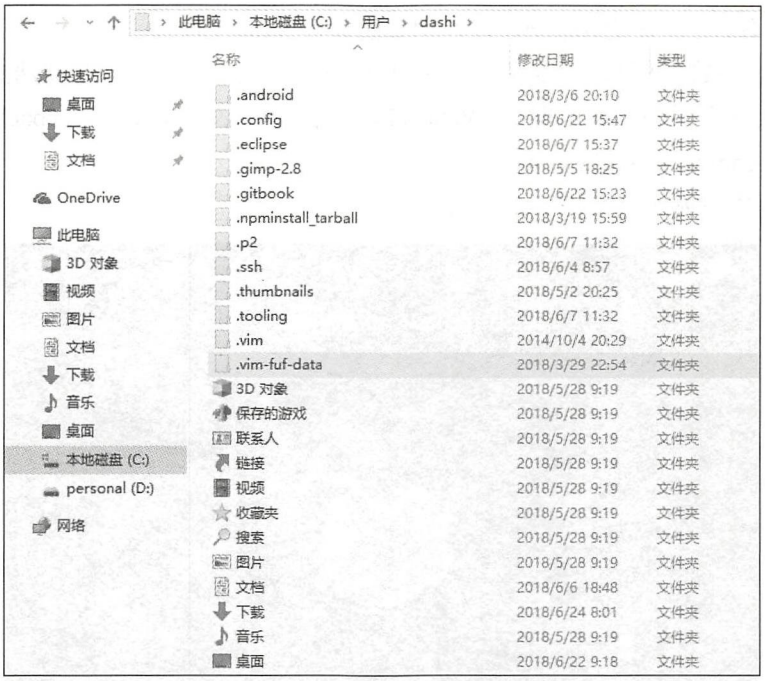


图 3-21　用户文件夹

可以看到输入的路径是 C:\Users\dashi，结果在 GUI 中显示的文字是 "> 此电脑 > 本地磁盘(C:) > 用户 > dashi"。

（3）切换路径：cd。

例如，想进入工作目录（位于 D:\workspace\happy_book_Vue.js），继续写关于 Vue.js 书，就可以这样：

```
dashi@i5-16g MINGW64 ~
$ cd /d/workspace/happy_book_Vue.js/

dashi@i5-16g MINGW64 /d/workspace/happy_book_Vue.js (master)
$
```

大家可以看到，D:\在 Git Bash 中对应的地址是/d，这个就是唯一需要注意的点了。

其他 git 基本知识（git clone、git commit、git push 等）就不在本书中赘述了。

3.4 Webpack

随着 SPA（Single Page App）单页应用的发展，可以发现，使用的 js/css/png 等文件特别多，比较难管理，文件夹结构也很容易混乱。很多时候我们希望项目可以具备压缩 css，压缩

js，正确地处理各种 js/css 的 import，以及相关的模板 html 文件。

在最开始的一段时间里，可以说每个 SPA 项目发展到一定规模，都会遇到这样的瓶颈和痛点。为了解决这个问题，就出现了 Webpack，其官方网站为 https://webpack.js.org/，github 为 https://github.com/webpack/ webpack。

Webpack 官方网站页面如图 3-22 所示。

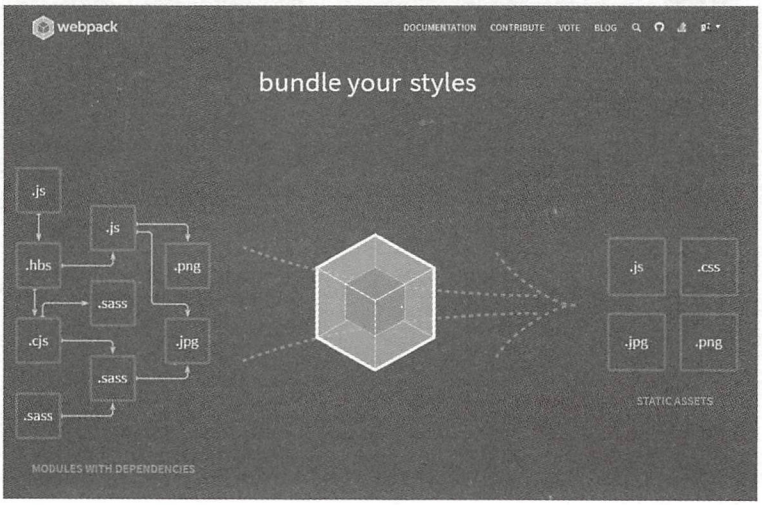


图 3-22　Webpack 官方页面

Webpack 是一个打包工具，可以把 js、css、node module、coffeescrip、scss/less、图片等都打包在一起，简直是模块化开发 SPA 福音。因此，现在几乎所有的 SPA 项目、JS 项目都会用到 Webpack。

在前面的入门知识中，我们看到 Vue.js 只需要引入一个外部的 js 文件就可以工作了。不过，在实际开发中，情况就复杂了很多倍，我们都是统一使用 Webpack + Vue.js 的方式来做项目的，这样才可以做到“视图”“路由”“component”等的分离，以及快速打包、部署及项目上线。

3.4.1　Webpack 功能

Webpack 功能非常强大，对各种技术都提供了支持，仿佛是一个“万能胶水”，把所有的技术都结合到了一起。

1. 对文件的支持

- 支持普通文件
- 支持代码文件
- 支持文件转 url（支持图片）

2. 对 JSON 的支持

- 支持普通 JSON

- 支持 JSON5
- 支持 CSON

3. 对 JS 预处理器的支持

- 支持普通 JavaScript
- 支持 Babel（使用 ES2015+）
- 支持 Traceur（使用 ES2015+）
- 支持 Typescript
- 支持 Coffeescript

4. 对模板的支持

- 支持普通 HTML
- 支持 Pug 模板
- 支持 JADE 模板
- 支持 Markdown 模板
- 支持 PostHTML
- 支持 Handlebars

5. 对 Style 的支持

- 支持普通 style
- 支持 import
- 支持 LESS
- 支持 SASS/SCSS
- 支持 Stylus
- 支持 PostCSS

6. 对各种框架的支持

- 支持 Vue.js
- 支持 Angular2
- 支持 Riot

3.4.2　Webpack 安装与使用

Webpack 安装命令如下：

```
$ npm install --save-dev webpack
```

因为 Webpack 自身是支持 Vue.js 的，所以 Webpack 与 Vue.js 已经结合到很难分清谁是谁，我们就索性不区分。知道做什么事情需要运行什么命令就可以了。

在接下来内容中，读者会看到 Webpack+Vue.js 共同开发的方法和步骤。

3.5 开发环境的搭建

- NPM：3.10.8 (NPM > 3.0)
- Node：v6.9.1 (Node > 6.0)
- Vue.js：2.0+

总体来说，只要能安装上 Node 和 Vue.js 就可以。

3.5.1 安装 Vue.js

需要同时安装 vue 和 vue-cli 这两个 node package。

运行下面的命令：

```
$ npm install vue vue-cli -g
```

`-g` 表示这个包安装后可以被全局使用。

3.5.2 运行 vue

创建一个基于 Webpack 模板的新项目。

```
$ vue init webpack my-project
```

注意，我们使用 Vue 都是在 Webpack 前提下使用的。

安装依赖：

```
$ cd my-project
$ cnpm install
```

在本地以默认端口来运行。

```
$ npm run dev
```

然后就可以看到在本地已经运行起来了。

```
> test_vue_0613@1.0.0 dev /workspace/test_vue_0613
> node build/dev-server.js

> Starting dev server...


 DONE  Compiled successfully in 12611ms   12:16:47 PM
```

```
> Listening at http://localhost:8080
```

打开 http://localhost:8080 就可以看到刚才创建的项目欢迎页，如图 3-23 所示。

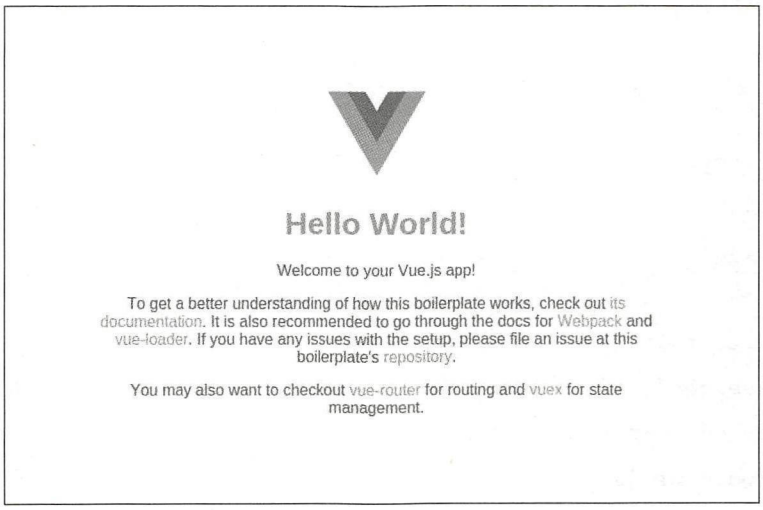


图 3-23　项目欢迎页

3.6 Webpack 下的 Vue.js 项目文件结构

前面我们已经安装了 Webpack、vue-cli 并运行成功，看到了 Vue.js 的第一个页面。那么我们首先需要对 Webpack 下的 Vue.js 有一个全面的了解。

运行下面命令后：

```
$ vue init webpack my-project
```

会生成一个崭新的项目。它的文件结构如下：

```
▸ build/                  // 编译用到的脚本
  config/                 // 各种配置
  dist/                   // 打包后的文件夹
  node_modules/           // node 第三方包
  src/                    // 源代码
  static/                 // 静态文件，暂时无用
  index.html              // 最外层文件
  package.json            // node 项目配置文件
```

下面我们针对重要的文件和文件夹依次说明。

3.6.1 build 文件夹

build 文件夹中保留了各种打包脚本，是不可或缺的，不可随意修改。

build 文件夹展开后如下：

```
▾ build/
    build.js
    check-versions.js
    dev-client.js
    dev-server.js
    utils.js
    vue-loader.conf.js
    webpack.base.conf.js
    webpack.dev.conf.js
    webpack.prod.conf.js
```

- build.js：打包使用，不能修改。
- check-versions.js：检查 npm 的版本，不能修改。
- dev-client.js 和 dev-server.js：是在开发时使用的服务器脚本，不能修改（借助于 node 后端语言，在做 Vue.js 开发时，可以通过$npm run dev 命令打开一个小的 server 运行 Vue.js）。
- utils.js：不能修改，做一些 css/sass 等文件的生成。
- vue-loader.conf.js：非常重要的配置文件，不能修改。内容是用于辅助加载 Vue.js 用到的 css source map 等。
- Webpack.base.conf.js、Webpack.dev.conf.js、Webpack.prod.conf.js：这三个都是基本的配置文件，不能修改。

3.6.2 config 文件夹

与部署和配置相关。

```
▾ config/
    dev.env.js
    index.js
    prod.env.js
```

- dev.env.js：开发模式下的配置文件，一般不用修改。
- prod.env.js：生产模式下的配置文件，一般不用修改。
- index.js：很重要的文件，定义了开发时的端口（默认 8080）、图片文件夹（默认 static）、开发模式下的代理服务器。

对于这个文件夹的内容，会在后续的章节中陆续进行解释（如对于某个页面的渲染过程、如何做代理转发）。

3.6.3　dist 文件夹

打包之后的文件所在目录如下：

```
▾ dist/
    static/
      css/
        app.d41d8cd98f00b204e9800998ecf8427e.css
        app.d41d8cd98f00b204e9800998ecf8427e.css.map
      js/
        app.c482246388114c3b9cf0.js
        app.c482246388114c3b9cf0.js.map
        manifest.577e472792d533aaaf04.js
        manifest.577e472792d533aaaf04.js.map
        vendor.5f34d51c868c93d3fb31.js
        vendor.5f34d51c868c93d3fb31.js.map
    index.html
```

可以看到，对应的 css、js、map 都在这里。

注意，文件名中无意义的字符串是随机生成的。目的是为了让文件名发生变化，方便部署，同时方便 Nginx 服务器重新对该文件进行缓存。

- app.css：编译后的 CSS 文件。
- app.js：最核心的 js 文件，几乎所有的代码逻辑都会打包到这里。
- manifest.js：生成的周边 js 文件。
- vendor.js：生成的 vendor.js 文件。

另外，每个.map 文件都非常重要。可以简单地认为，有了.map 文件，浏览器就可以先下载整个.js 文件，然后在后续的操作中“部分加载”对应的文件。

切记这个文件夹不要放到 git 中。因为每次编译之后，这里的文件都会发生变化。

3.6.4　node_modules 文件夹

node 项目所用到的第三方包特别多，也特别大。这些文件是由命令$ npm install 产生的。所有在 package.json 中定义的第三方包都会被自动下载，保存在该文件夹下。

package.json：

```
// ...
```

```
"dependencies": {
  "vue": "^2.3.3",
  "vue-resource": "1.3.3",
  "vue-router": "^2.3.1",
  "vuex": "^2.3.1"
},
"devDependencies": {
  "autoprefixer": "^6.7.2",
  "babel-core": "^6.22.1",
  "babel-loader": "^6.2.10",
  "babel-plugin-transform-runtime": "^6.22.0",
  "babel-preset-env": "^1.3.2",
  "babel-preset-stage-2": "^6.22.0",
  "babel-register": "^6.22.0",
  //...
}
// ...
```

node_modules 文件夹中往往会有几百个文件夹，如图 3-24 所示。

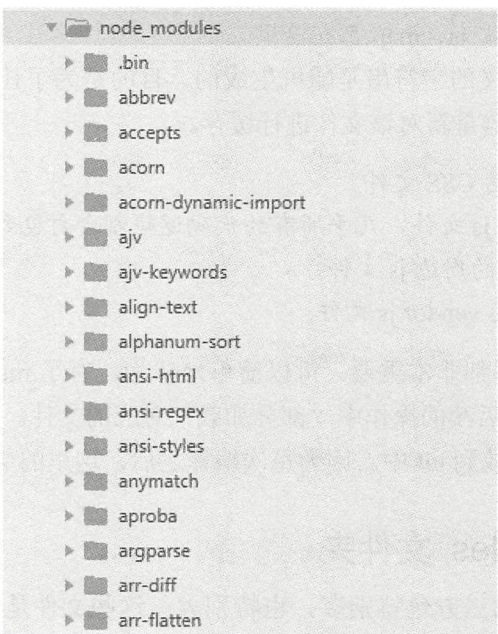
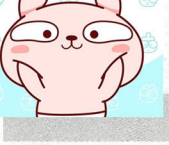


图 3-24　node_modules 文件夹

注意，这个文件夹不能放到 git 中。



3.6.5　src 文件夹

src 文件夹是核心源代码的所在目录。展开后如下所示（不同版本的 vue-cli 生成的目录会稍有不同，不过核心都是一样的）：

```
▾ src/
    assets/
      logo.png
    components/
      Book.vue
      BookList.vue
      Hello.vue
    router/
      index.js
    App.vue
    main.js
```

- assets 文件夹用到的图片都可以放在这里。
- components 用到的“视图”和“组件”所在的文件夹（核心）。
- router/index.js 是路由文件，定义了各个页面对应的 url。
- 如果 index.html 是一级页面模板的话，App.vue 就是二级页面模板。所有的其他 Vue.js 页面都作为该模板的一部分被渲染出来。
- main.js 没有实际的业务逻辑，但是为了支撑整个 Vue.js 框架，很有必要存在。

第 4 章

Webpack+Vue.js实战

4.1 创建一个页面

激动人心的时刻到来了，接下来我们需要通过自己动手开始下一步的学习。

请读者务必准备好电脑，只有一边学习一边编写代码，才能真正看到效果，因为调试代码的过程是无法脑补出来的。

在 Vue.js 中创建页面需要以下两步。

（1）新建路由。

（2）新建 vue 页面。

4.1.1 新建路由

默认的路由文件是 src/router/index.js，将其打开之后，我们增加两行：

```
import Vue from 'vue'
import Router from 'vue-router'

// 增加这一行，作用是引入 SayHi 这个 component
import SayHi from '@/components/SayHi'

Vue.use(Router)
export default new Router({
  routes: [

    // 增加下面几行，表示定义了  /#/say_hi 这个 url
    {
```

```
    path: '/say_hi',
    name: 'SayHi',
    component: SayHi
  },
 ]
})
```

上面的代码中：

```
import SayHi from '@/components/SayHi'
```

表示从当前目录下的 components 引入 SayHi 文件，@表示当前目录。

然后利用下面的代码定义一个路由：

```
routes: [
  {
    path: '/say_hi',   // 对应一个 url
    name: 'SayHi',     // Vue.js 内部使用的名称
    component: SayHi   // 对应的.vue 页面的名字
  },
]
```

也就是说，每当用户的浏览器访问 http://localhost:8080/#/say_hi 时，就会渲染./components/SayHi.vue 文件。name: 'SayHi' 定义了该路由在 Vue.js 内部的名称。

4.1.2　创建一个新的 Component

由于我们在路由中引用了 component：src/components/SayHi.vue，接下来，就创建这个文件。代码如下：

```
<template>
  <div >
    Hi Vue!
  </div>
</template>

<script>
export default {
  data () {
    return { }
```

```
  }
}
</script>

<style>
</style>
```

在上面的代码中：

- <template></template>代码块中表示的是 HTML 模板，里面写的就是最普通的 HTML 。
- <script/>表示的是 js 代码，所有的 js 代码都写在这里。这里使用的是 EMScript。
- <style>表示所有的 CSS/SCSS/SASS 文件都可以写在这里。

现在，可以直接访问 http://localhost:8080/#/say_hi 了，页面如图 4-1 所示。

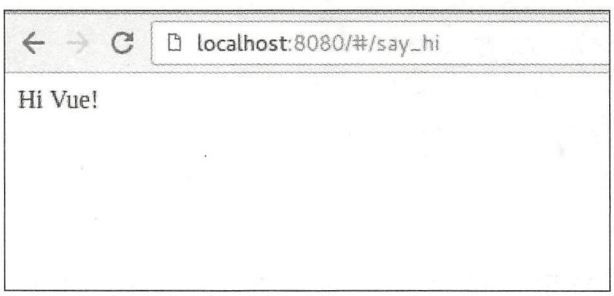


图 4-1　页面效果

4.1.3　为页面添加样式

我们可以为页面添加一些样式，让它变得好看一些。

```
<template>
  <div class='hi'>
    Hi Vue!
  </div>
</template>

<script>
export default {
  data () {
    return { }
  }
```

```
}
</script>

<style>
.hi {
  color: red;
  font-size: 20px;
}
</style>
```

注意上面代码中的<style>标签，里面与普通的 CSS 一样定义了样式。

```
.hi {
  color: red;
  font-size: 20px;
}
```

刷新浏览器，可以看到文字加了颜色，如图 4-2 所示。

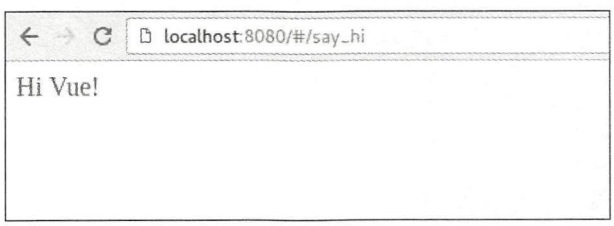


图 4-2　为文字添加颜色

4.1.4　定义并显示变量

如果要在 vue 页面中定义一个变量，并显示出来，就需要事先在 data 中定义。

```
export default {
  data () {
    return {
      message: '你好 Vue! 本消息来自于变量'
    }
  }
}
```

可以看到，上面的代码是通过 Webpack 的项目来写的。回忆一下，在原生的 Vue.js 中是如何定义一个变量（property）的？

答案是：

```
var app = new Vue({
  data () {
    return {
      message: '你好 Vue! 本消息来自于变量'
    }
  }
})
```

我们可以认为，之前在“原生的 Vue.js”的代码中存在于 new Vue({...})中的代码，在 Webpack 框架下，都应该放到 export default{ .. }代码块中。

完整的代码（src/components/SayHi.vue）如下所示：

```
<template>
  <div>
    <!-- 步骤 2: 在这里显示 message -->

  </div>
</template>

<script>
export default {
  data () {
    return {
      // 步骤 1: 这里定义了变量 message 的初始值
      message: '你好 Vue! 本消息来自于变量'
    }
  }
}
</script>

<style>
</style>
```

页面效果如图 4-3 所示。

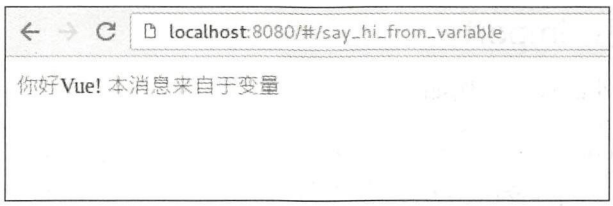


图 4-3　页面效果

4.2 Vue.js 中的 ECMAScript

有一定编程经验的读者会发现，我们使用的不是“原生的 JavaScript”，而是一种新的语言，这个语言就是 ECMAScript。

严格来说，ECMAScript 是 JavaScript 的规范，JavaScript 是 ECMA 的实现。

ECMAScript 的简称是 ES，其版本比较多，有 ES 2015、ES 2016、ES 2017 等，很多时候我们用 ES6 来泛指这三个版本。从 http://kangax.github.io/compat-table/es6/ 可以看到，ES6 的 90%的特性都已经被各大浏览器实现了。

具体的细节不去深究，我们就暂且认为 ECMAScript 实现了很多普通 js 无法实现的功能。同时，在 Vue.js 项目中大量使用了 ES 的语法。

下面是极简版的 ES6 入门知识。大家只要看懂这些代码，就可以继续阅读本书。

4.2.1 let、var、常量与全局变量

声明本地变量，使用 let 或 var，两者的区别如下。

- var：有可能引起变量提升，或者块级作用域的问题。
- let：就是为了解决以上两个问题存在的。

最佳实践：多用 let，少用 var，遇到诡异变量问题时，就查一查是不是 var 的问题。下面是三个对比：

```
var title = '标题';   // 没问题
let title = '标题';   // 没问题
title = '标题';       // 这样做会报错.
```

在 Webpack 下的 Vue.js 中使用任何变量，都要使用 var 或 let 来声明。常量：

```
const TITLE='标题';
```

对于全局变量，直接在 index.html 中声明即可。例如：

```
window.title = '我的博客列表'
```

4.2.2 导入代码：import

import 用于导入外部代码。例如：

```
import Vue from 'vue'
import Router from 'vue-router'
```

上面的代码，目的是引入 vue 和 vue-router（由于它们是在 package.json 中定义的，因此可以直接 import ... from <包名>，否则要加上路径）。

```
import SayHi from '@/components/SayHi'
```

在 from 后面添加@符号，表示是在本地文件系统中引入文件。@代表源代码目录，一般是 src。

@出现之前，我们在编码时也会这样写：

```
import Swiper from '../components/swiper'
import SwiperItem from '../components/swiper-item'
import XHeader from '../components/header/x-header'
import store from '../vuex/store'
```

因为大量使用了../..，这样的代码会引起混乱，所以推荐使用@方法。

4.2.3 方便其他代码使用自己: export default {..}

在每个 vue 文件的<script>中，都会存在 export default {..}代码，作用是方便其他代码对这个代码进行引用。对于 Vue.js 程序员来说，记住这个写法就可以了。

在 ES6 之前，js 没有统一的模块定义方式，流行的定义方式有 AMD、CommonJS 等，这些方式都是以一种“打补丁”的形式实现这个功能。ES6 从语言层面对定义模块的方式进行了统一。

假设有 lib/math.js 文件，其内容如下：

```
export function sum(x, y) {
  return x + y
}
export var pi = 3.141593
```

lib/math.js 文件可以导出两个内容，一个是 function sum，另一个是 var pi。

我们可以定义一个新的文件：app.js，内容如下：

```
import * as math from "lib/math"
alert("2π = " + math.sum(math.pi, math.pi))
```

在上面的代码中， 可以直接调用 math.sum 和 math.pi 方法。

新建一个文件：other_app.js，内容如下：

```
import {sum, pi} from "lib/math"
alert("2π = " + sum(pi, pi))
```

在上面的代码中，通过 import {sum, pi} from "lib/math"可以在后面直接调用 sum()和 pi。而 export default { ... } 则是暴露出一段没有名字的代码，不像 export function sum(a,b){ .. }有一个名字（sum）。

在 Webpack 下的 Vue.js，会自动对这些代码进行处理，属于框架内的工作。读者只要按照这个规则来写代码，就一定没有问题。

4.2.4　ES 中的简写

有时我们会发现这样的代码：

```
<script>
export default {
  data () {
    return { }
  }
}
</script>
```

实际上，上面的代码是一种简写形式，等同于下面的代码：

```
<script>
export default {
  data: function() {
    return { }
  }
}
</script>
```

4.2.5　箭头函数=>

与 coffeescript 一样，ES 也可以通过箭头表示函数。

```
  .then(response => ... );
```

等同于：

```
  .then(function (response) {
    // ...
```

```
})
```

这样写的好处就是强制定义了作用域。使用=>之后，可以避免很多由作用域产生的问题，建议大家多使用。

4.2.6 hash 中同名的 key、value 的简写

```
let title = 'triple body'

return {
  title: title
}
```

等同于：

```
let title = 'triple body'

return {
  title
}
```

4.2.7 分号可以省略

例如：

```
var a = 1
var b = 2
```

等同于：

```
var a = 1;
var b = 2;
```

4.2.8 解构赋值

我们先定义好一个 hash：

```
let person = {
  firstname : "steve",
  lastname : "curry",
  age : 29,
  sex : "man"
};
```

然后可以这样做定义：

```
let {firstname, lastname} = person
```

上面一行代码等同于：

```
let firstname = person.firstname
let lastname = person.lastname
```

可以这样定义函数：

```
function greet({firstname, lastname}) {
  console.log(`hello,${firstname}.${lastname}!`);
};

greet({
  firstname: 'steve',
  lastname: 'curry'
});
```

但是不建议读者这样使用。另外，浏览器和一些第三方支持应用的不是太好，我们在实际项目中，曾经遇到过与之相关的很奇葩的问题。

注　意

有关 ECMAScript，国内比较好的中文教材，是阮一峰编写的 《ES6 标准入门（第 3 版）》，书中非常详实地阐述了相关的内容、概念和用法，也可以在网上直接查看这本书的电子版：http://es6.ruanyifeng.com。

4.3 Vue.js 渲染页面的过程和原理

只有知道一个页面是如何被渲染出来的，才能更好地理解框架和调试代码。下面就来学习一下这个过程。

4.3.1 渲染过程 1：js 入口文件

首先我们要知道./build/webpack.base.conf.js 是 webpack 打包的主要配置文件。一个典型的代码如下：

```
var path = require('path')
var utils = require('./utils')
```

```
var config = require('../config')
var vueLoaderConfig = require('./vue-loader.conf')

function resolve (dir) {
  return path.join(__dirname, '..', dir)
}

module.exports = {
  entry: {
    app: './src/main.js'
  },
  output: {
    path: config.build.assetsRoot,
    filename: '[name].js',
    publicPath: process.env.NODE_ENV === 'production'
      ? config.build.assetsPublicPath
      : config.dev.assetsPublicPath
  },
  resolve: {
    extensions: ['.js', '.vue', '.json'],
    alias: {
      'vue$': 'vue/dist/vue.esm.js',
      '@': resolve('src')
    }
  },
  module: {
    rules: [
      {
        test: /\.vue$/,
        loader: 'vue-loader',
        options: vueLoaderConfig
      },
      {
        test: /\.js$/,
        loader: 'babel-loader',
        include: [resolve('src'), resolve('test')]
```

```
      },
      {
        test: /\.(png|jpe?g|gif|svg)(\?.*)?$/,
        loader: 'url-loader',
        options: {
          limit: 10000,
          name: utils.assetsPath('img/[name].[hash:7].[ext]')
        }
      },
      {
        test: /\.(woff2?|eot|ttf|otf)(\?.*)?$/,
        loader: 'url-loader',
        options: {
          limit: 10000,
          name: utils.assetsPath('fonts/[name].[hash:7].[ext]')
        }
      }
    ]
  }
}
```

在上面的代码中：

```
module.exports = {
  entry : {
    app: './src/main.js'   // 这里就定义了 Vue.js 的入口文件
  }
}
```

其中 app：'./src/main.js'就定义了 Vue.js 的入口文件。

4.3.2　渲染过程 2：静态的 HTML 页面（index.html）

因为我们每次打开的都是'http://localhost/#/'，实际上打开的文件是'http://localhost/#/index.html'，所以找到 index.html，就可以看到其内容。

```
<!DOCTYPE html>
<html>
  <head>
```


```
    <meta charset="utf-8">
    <title>演示 Vue 的 demo</title>
  </head>
  <body>
    <div id="app"></div>
  </body>
</html>
```

这里的 就是将来会动态变化的内容。特别类似于 Rails 的 layout（模板文件）。

这个 index.html 文件是最外层的模板。

4.3.3 渲染过程 3：main.js 中的 Vue 定义

我们来看看 src/main.js 文件，其内容如下：

```
import Vue from 'vue'
import App from './App'
import router from './router'
import VueResource from 'vue-resource';
Vue.use(VueResource);

Vue.config.productionTip = false
Vue.http.options.emulateJSON = true;

new Vue({
  el: '#app',   // 这里, #app 对应着 <div id=app></div>
  router,
  template: '<App/>',
  components: { App }  // 这里, App 就是 ./src/App.vue
})
```

熟悉 jquery、css selector 的读者可以看到，el: '#app'就是 index.html 中的

上面的 App.vue 会被 main.js 加载。App.vue 的内容如下：

```
<template>
  <div id="app">
      <router-view class="view"></router-view>
  </div>
```

```
</template>
<script>
</script>
<style>
</style>
```

上面代码中的<template/>就是第二层模板，可以认为该页面的内容就是在这个位置被渲染出来的。

在上面的代码中，还有 ... ，该元素与 index.html 中的是同一个元素（暂且这样理解）。

所有<router-view>中的内容都会被自动替换。<script>中的代码则是脚本代码。至此，整个过程就出来了。

4.3.4　渲染原理与实例

Vue.js 就是最典型的 Ajax 工作方式，即只渲染部分页面。

浏览器的页面从来不会整体刷新，所有的页面变化都限定在 index.html 中的 代码中。

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>演示 Vue 的 demo</title>
  </head>
  <body>
    <div id="app"></div>   <!-- 都在这一行 -->
  </body>
</html>
```

所有的动作都可以靠 url 来触发。例如：

- /#/books_list 对应某个列表页。
- /#/books/3 对应某个详情页。

这个技术就是靠 Vue.js 的核心组件 vue-router 来实现的。

不使用 router 的技术：QQ 邮箱

QQ 邮箱是属于 url 无法与某个页面一一对应的项目。所有页面的跳转，都无法根据 url 来判断。

最大的特点是不能保存页面的状态，难以调试，无法根据 url 进入某个页面。

4.4 视图中的渲染

前面我们介绍了项目的运行（hello world）、文件夹的结构及 index.html 中的内容是如何一点点渲染出来的。下面学习 Vue.js 中视图的操作。

4.4.1 渲染某个变量

假设定义了一个变量：

```
<script>
export default {
  data () {
    return {
      my_value: '默认值',
    }
  },
}
</script>
```

可以这样来显示它：

```
<div></div>
```

完整代码如下：（src/components/Hello.vue）

```
<template>
  <div>

  </div>
</template>

<script>
export default {
  data () {
    return {
      message: '你好 Vue! 本消息来自于变量'
```

```
    }
  }
}
</script>

<style>
</style>
```

上面的代码显示定义了 message 变量，然后将其在<h1> </h1>中显示出来。

打开 http://localhost:8080/#/say_hi_from_variable 页面，就可以看到如图 4-4 所示的结果。

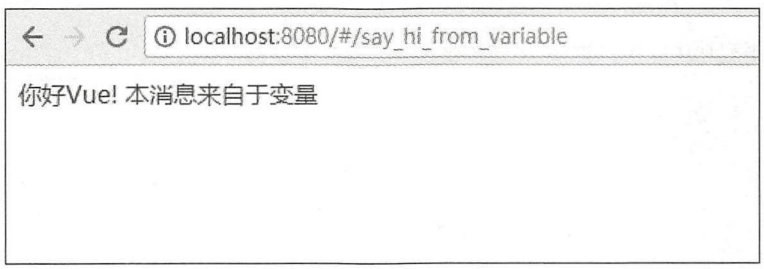


图 4-4　运行结果

4.4.2　方法的声明和调用

声明一个 show_my_value 方法。

```
<script>
export default {
  data () {
    return {
      my_value: '默认值',
    }
  },
  methods: {
    show_my_value: function(){
      // 注意下面的  this.my_value, 要用到 this 关键字
      alert('my_value: ' + this.my_value);
    },
  }
}
</script>
```

调用上面的方法。

```
<template>
  <div>
    <input type='button' @click="show_my_value()" value='...'/>
  </div>
</template>
```

对于有参数的方法，直接传递参数就可以了。例如：

```
<template>
  <div>
    <input type='button' @click="say_hi('Jim')" value='...'/>
  </div>
</template>
<script>
export default {
  data () {
    return {
      my_value: '默认值',
    }
  },
  methods: {
    say_hi: function(name){
      alert('hi, ' + name)
    },
  }
}
</script>
```

上面的代码中：

```
<input type='button' @click="say_hi('Jim')" value='...'/>
```

就会调用 say_hi 方法，传入参数'Jim'。

4.4.3 事件处理：v-on

很多时候，@click 等同于 v-on:click。下面两个是一样的：

```
<input type='button' @click="say_hi('Jim')" value='...'/>
```


```
<input type='button' v-on:click="say_hi('Jim')" value='...'/>
```

4.5 视图中的 Directive（指令）

我们在学习 Java 的时候，知道有 jsp 页面，对于.net 语言，有.asp、aspx 页面， 对于 Ruby，有 erb 页面，在 Vue.js 中也有类似的编程能力。但是因为 Vue.js 是一种 javascript 框架，所以只能与标签结合使用，叫做 Directive（指令）。

我们之前看到的 v-on、v-bind、v-if、v-for 等，只要是以 v 开头的，都是 Directive。

原理：

（1）用户在浏览器中输入网址，按回车键。

（2）浏览器加载所有的资源（js、html 内容），此时尚未解析。

（3）浏览器加载 Vue.js。

（4）Vue.js 程序被执行，发现页面中的 Directive 并进行相关的解析。

（5）HTML 中的内容被替换，展现给用户。

因此，我们在开发一个 Vue.js 项目的时候，会看到大量的 Directive。这里的基础务必打好。

4.5.1 前提：在 directive 中使用表达式（Expression）

- 表达式：a>1（有效）。
- 普通语句：a=1;（这是个声明，不会生效）。
- 控制语句：returna;（不会生效）。

在所有的 Directive 中，只能使用表达式。

正确的：

```
<div v-if="a > 100">
</div>
```

错误的：

```
<div v-if="return false">
</div>
```

4.5.2 循环：v-for

完整的代码如下：

```
<html>
<head>
    <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
</head>
<body>
    <div id='app'>
        <p>Vue.js 周边的技术生态有： <p>
        <br/>
        <ul>
            <li v-for="tech in technologies">
                {{tech}}
            </li>
        </ul>
    </div>
    <script>
        var app = new Vue({
            el: '#app',
            data: {
                technologies: [
                    "nvm", "npm", "node", "webpack", "ecma_script"
                ]
            }
        })
    </script>
</body>
</html>
```

上面代码中的 technologies 是在 data 中被定义的。

```
data: {
    technologies: [
        "nvm", "npm", "node", "webpack", "ecma_script"
    ]
}
```

然后在下面的代码中被循环显示。

```
<li v-for="tech in technologies">
```

```
</li>
```

使用浏览器打开上面代码后，结果如图 4-5 所示。

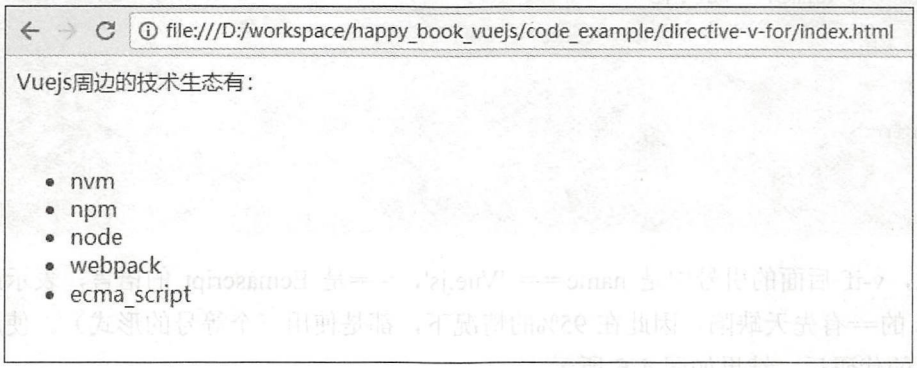


图 4-5　运行结果

4.5.3　判断：v-if

条件 Directive 是由 v-if、v-else-if、v-else 配合完成的。下面是一个完整的例子：

```
<html>
<head>
    <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
</head>
<body>
    <div id='app'>
        <p>我们要使用的技术是： <p>

        <div v-if="name === 'Vue.js'">
            Vue.js !
        </div>
        <div v-else-if="name === 'angular'">
            Angular
        </div>
        <div v-else>
            React
        </div>
    </div>
    <script>
```

```
        var app = new Vue({
            el: '#app',
            data: {
                name: 'Vue.js'
            }
        })
    </script>
</body>
</html>
```

注意，v-if 后面的引号中是 name=== 'Vue.js'，===是 Ecmascript 的语言，表示严格判断（由于 js 的==有先天缺陷，因此在 95%的情况下，都是使用三个等号的形式）。使用浏览器打开上面的代码后，结果如图 4-6 所示。

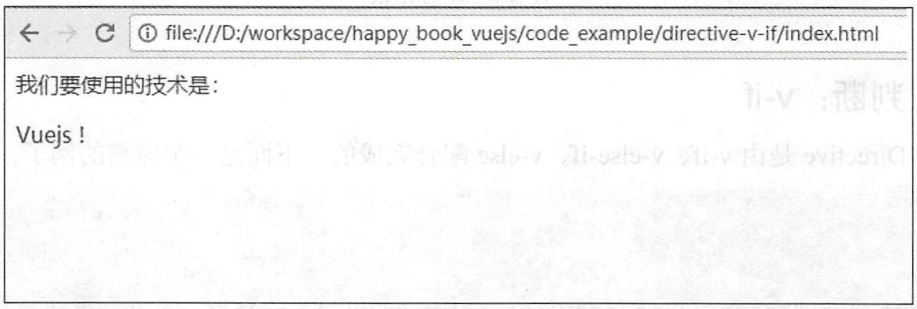


图 4-6　运行结果

4.5.4　v-if 与 v-for 的优先级

当 v-if 与 v-for 一起使用时，v-for 具有比 v-if 更高的优先级。也就是说，Vue.js 会先执行 v-for，再执行 v-if。

下面是个完整的例子：

```
<html>
<head>
    <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
</head>
<body>
    <div id='app'>
        <p> 全部的技术是:  </p>
        <p> v-for 与 v-if 的结合使用, 只打印出 以"n"开头的技术:  <p>
        <ul>
```

```
            <li v-for="tech in technologies" v-if="tech.indexOf('n') === 0">
                {{tech}}
            </li>
        </ul>
    </div>
    <script>
        var app = new Vue({
            el: '#app',
            data: {
                technologies: [
                    "nvm", "npm", "node", "webpack", "ecma_script"
                ]
            }
        })
    </script>
</body>
</html>
```

可以看到，在上面的代码中，v-if 与 v-for 结合使用了，先是做了循环 tech in technologies，然后对当前的循环对象 tech 做了判断。核心代码如下所示：

```
<li v-for="tech in technologies" v-if="tech.indexOf('n') === 0">
    {{tech}}
</li>
```

使用浏览器打开上面的代码后，结果如图 4-7 所示。

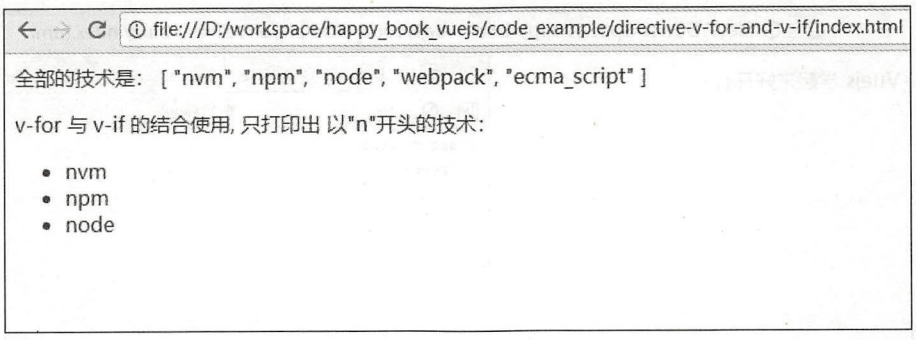


图 4-7　运行结果

4.5.5 v-bind

v-bind 指令用于把某个属性绑定到对应的元素属性。例如：

```
<html>
<head>
    <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
</head>
<body>
    <div id='app'>
        <p v-bind:style="'color:' + my_color">Vue.js 学起来好开心~ </p>
    </div>
    <script>
        var app = new Vue({
            el: '#app',
            data: {
                my_color: 'green'
            }
        })
    </script>
</body>
</html>
```

在上面的代码中，通过 v-bind 把<p>元素的 style 的值绑定成了'color:' + my_color 表达式。当 my_color 的值发生变化时，对应<p>的颜色也会发生变化。

例如：默认页面打开后，文字是绿色的，如图 4-8 所示。

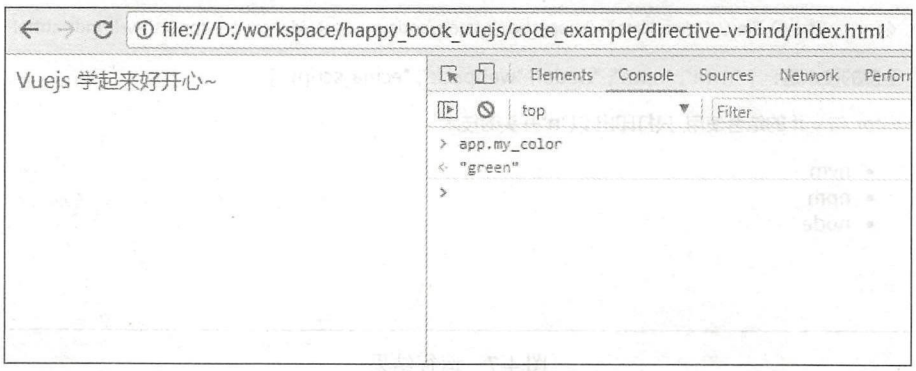


图 4-8 默认文字是绿色的

如何知道变量 my_color 已经绑定到<p>上了呢？我们在 console 中做修改，让 app.my_color = "red"，就可以看到对应的文字颜色变成了红色，如图 4-9 所示。

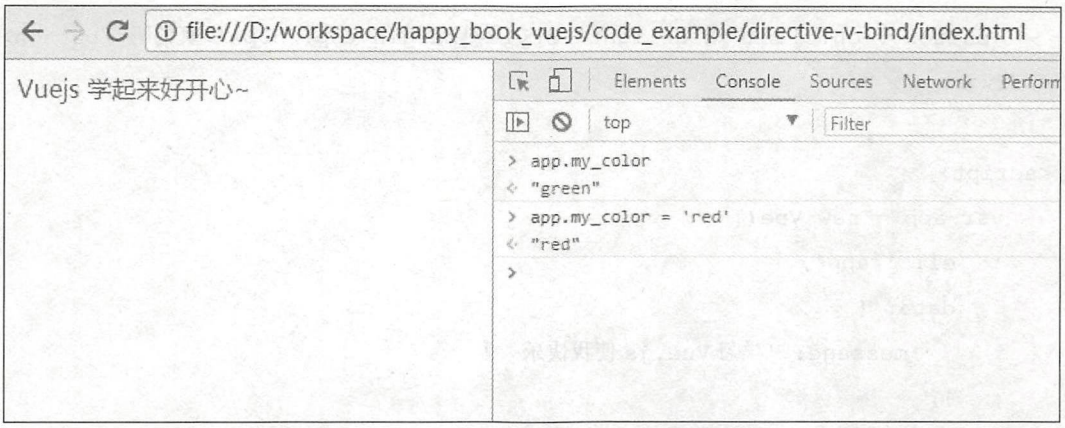


图 4-9　文字变成了红色

对于所有的属性，都可以使用 v-bind。例如：

```
<div v-bind:style='...'>
</div>
```

会生成：

```
<div style='...'>  </div>
<div v-bind:class='...'> </div>
```

会生成：

```
<div class='...'>  </div>
<div v-bind:id='...'> </div>
```

会生成：

```
<div id='...'>  </div>
```

4.5.6　v-on

v-on 指令用于触发事件。例如：

```
<html>
<head>
   <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
</head>
<body>
   <div id='app'>

      <br/>
```

```
        <button v-on:click='highlight' style='margin-top: 50px'>真的吗</button>
    </div>

    <script>
        var app = new Vue({
            el: '#app',
            data: {
                message: '学习 Vue.js 使我快乐~ '
            },
            methods: {
                highlight: function() {
                    this.message = this.message + '是的， 工资还会涨~!'
                }
            }

        })
    </script>
</body>
</html>
```

在上面的代码中，通过 v-on:click 的声明，当被单击（click）后，就会触发 highlight 方法。

单击前的页面如图 4-10 所示。

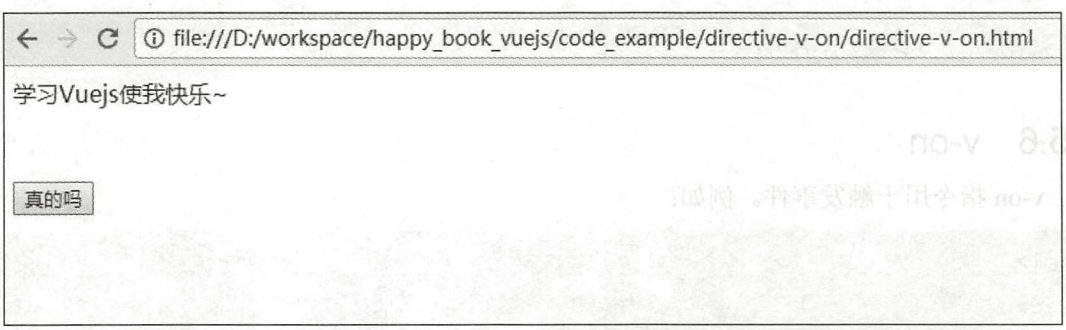


图 4-10　单击前的页面

单击后的页面如图 4-11 所示。

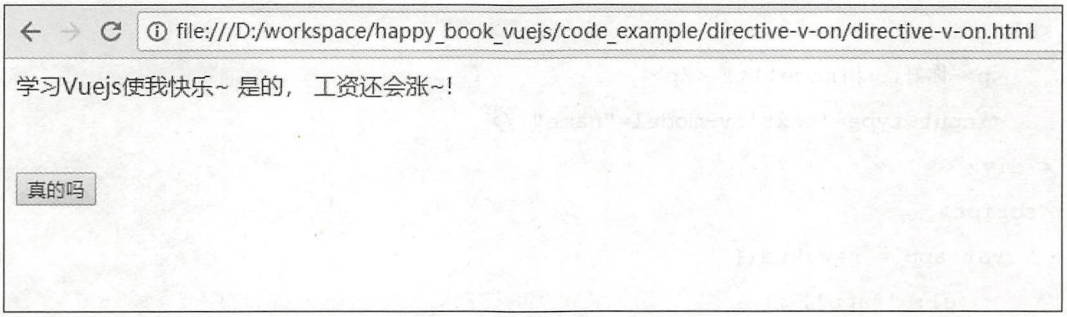


图 4-11　单击后的页面

v-on 后面可以接 HTML 的标准事件。例如：

- click（单击鼠标左键）
- dblclick（双击鼠标左键）
- contextmenu（单击鼠标右键）
- mouseover（指针移到有事件监听的元素或其子元素内）
- mouseout（指针移出元素，或者移到其子元素上）
- keydown（键盘动作：按下任意键）
- keyup（键盘动作：释放任意键）

对于 v-on 的更多说明，请参看 Event 的对应章节。

注意：v-on 可以简写，v-on:click 往往会写成@click，v-on:dblclick，也会写成@dblclick，读者看代码的时候要注意。

4.5.7　v-model 与双向绑定

v-model 往往用来做“双向绑定”（two way binding）。双向绑定的含义如下：

（1）可以通过表单（用户手动输入的值）来修改某个变量的值。

（2）可以通过程序的运算来修改某个变量的值，并且影响页面的展示。

双向绑定可以大大方便我们的开发。例如，在制作一个天气预报的软件时，需要在前台展示“当前温度”，如果后台代码做了某些操作后，就会发现“当前温度”发生了变化。

如果没有双向绑定，代码就会比较膨胀，做各种条件判断。有了双向绑定，就可以做到后台变量一变化，前台对于该变量的展示就会发生变化。

下面是一个完整的页面源代码。

```
<html>
<head>
   <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
</head>
<body>
```

```
    <div id='app'>
        <p> 你好， {{name}} ！ </p>
        <input type='text' v-model="name" />
    </div>
    <script>
        var app = new Vue({
            el: '#app',
            data: {
                // 下面一行代码不能省略。 这里声明了 name 变量
                name: 'Vue.js（默认）'
            }
        })
    </script>
</body>
</html>
```

在上面的代码中，可以看到：

（1）使用了 <input type='text' v-model="name" />把变量 name 绑定在<input>输入框上（可以在这里看到 name 的值）

（2）使用了 你好，{{name}} ! 把变量 name 显示在页面上。

（3）在初始化中，使用 data: { name: '...' } 的方式对变量 name 进行了初始化。

使用浏览器打开该 html 页面，可以看到最初的状态如图 4-12 所示。

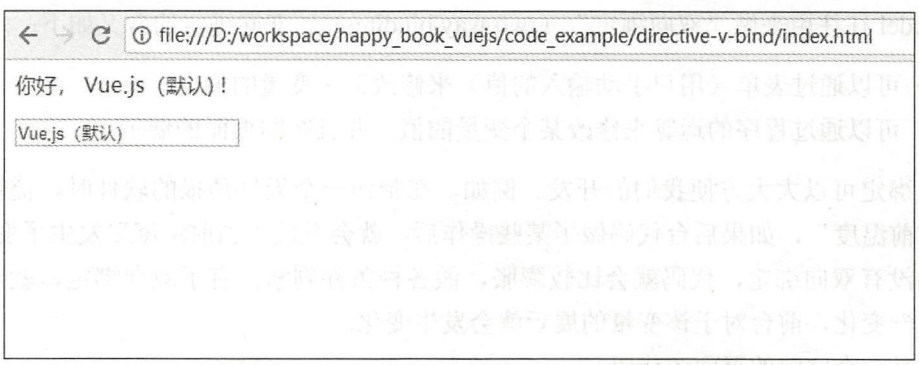


图 4-12　最初的状态

可以看到，图 4-12 中显示的文字是“你好，Vue.js!”。

然后在输入框中把内容改为“Vue.js 和 Webpack”，于是页面就发生了变化，如图 4-13 所示。

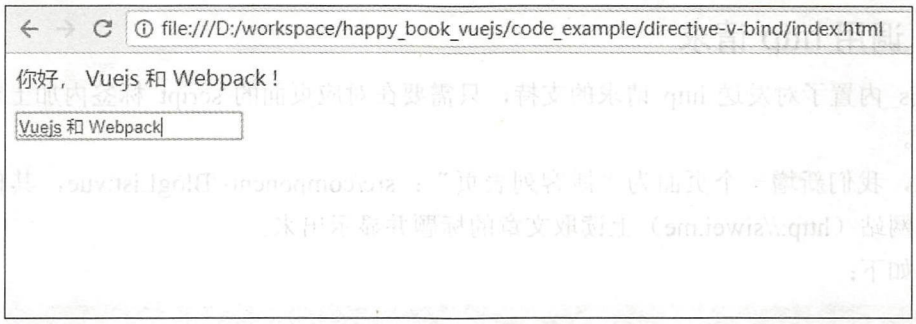


图 4-13　页面发生变化

这就说明，我们通过输入框来改变某个变量的值是成功的。

打开浏览器中的 Peveloper Tools（建议用 Chrome，Chrome 下的操作方式是按 F12 键）。在 console 中输入 app.name = "明日 Vue.js 高手"，就会看到如图 4-14 所示的效果。

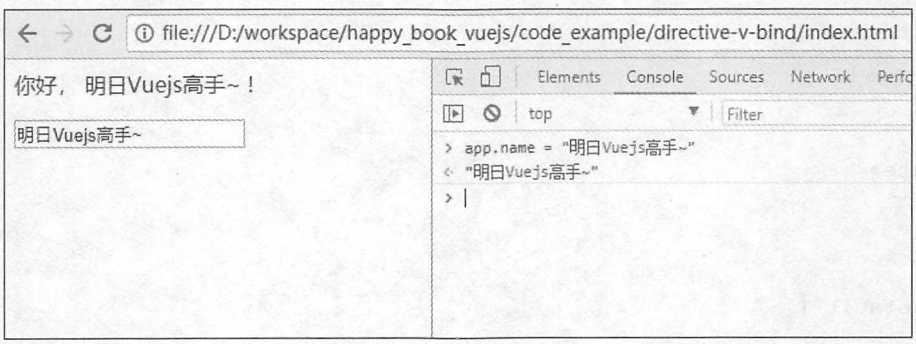


图 4-14　页面效果

这就说明，我们通过运算来改变某个变量的值是成功的。

4.6 发送 http 请求

每个 SPA 项目都要使用 http 请求，这些请求从服务器读取数据，然后:

（1）在前端页面进行展示，如论坛应用中显示文章列表。

（2）做一些逻辑判断，如注册页面需要判断某个用户名是否已经存在。

（3）做一些数据库的保存操作，如修改密码。

所以，http 请求非常重要。

在进行下面的学习之前，我们需要为当前的 SPA 项目加上 http 请求的支持。修改 src/main.js 文件，增加如下内容即可。

```
import VueResource from 'vue-resource';
Vue.use(VueResource);
```

4.6.1 调用 http 请求

Vue.js 内置了对发送 http 请求的支持，只需要在对应页面的 script 标签内加上对应的代码就可以。

例如，我们新增一个页面为“博客列表页”：src/components/BlogList.vue，其作用是从我的个人网站（http://siwei.me）上读取文章的标题并显示出来。

代码如下：

```
<template>
  <div >
    <table>
      <tr v-for="blog in blogs">
        <td></td>
      </tr>
    </table>
  </div>
</template>

<script>
export default {
  data () {
    return {
      title: '博客列表页',
      blogs: [
      ]
    }
  },
  mounted() {
    this.$http.get('api/interface/blogs/all').then((response) => {
      console.info(response.body)
      this.blogs = response.body.blogs
    }, (response) => {
      console.error(response)
    });
  }
}
</script>
```

```
<style >

td {
  border-bottom: 1px solid grey;
}
</style>
```

在上面的代码中， 我们先看 <script/>代码段：

```
export default {
  data () {
    return {
      title: '博客列表页',
      blogs: [
      ]
    }
  },
  mounted() {
    this.$http.get('api/interface/blogs/all').then((response) => {
      console.info(response.body)
      this.blogs = response.body.blogs
    }, (response) => {
      console.error(response)
    });
  }
}
```

上面的代码先定义了两个变量：title 和 blogs，然后定义了一个 mounted 方法。该方法表示当页面加载完毕后应该做哪些事情，是一个钩子方法。

```
this.$http.get('api/interface/blogs/all').then((response) => {
   console.info(response.body)
   this.blogs = response.body.blogs
}, (response) => {
   console.error(response)
});
```

上面的代码是发起 http 请求的核心代码。访问的接口地址是 api/interface/blogs/all，然后

使用 then 方法做下一步的事情， then 方法接受两个函数作为参数，第一个是成功后做什么，第二个是失败后做什么。

成功后的代码如下：

```
this.blogs = response.body.blogs
```

然后在对应的视图部分显示：

```
<tr v-for="blog in blogs">
  <td></td>
</tr>
```

4.6.2 远程接口的格式

在远程服务器上读取个人博客标题的接口已经提前做好了，是 http://siwei.me/interface/blogs/all。其内容如下：

```
{
  blogs: [
    {
      id: 1516,
      title: "网络安全资源",
      created_at: "2018-06-24T09:36:20+08:00"
    },
    {
      id: 1515,
      title: "github - 邀请伙伴后，需要修改权限",
      created_at: "2018-06-20T15:03:33+08:00"
    },
    {
      id: 1514,
      title: "ruby/rails - 根据浏览器的语言自动识别",
      created_at: "2018-06-19T08:28:44+08:00"
    },
    {
      id: 1513,
      title: "google cloud - 申请 VM 的经验",
      created_at: "2018-06-09T16:42:08+08:00"
    },
    {
      id: 1512,
      title: "验证码 - 使用 geetest 或网易云盾提供的动态二维码",
```

```
      created_at: "2018-06-07T09:28:14+08:00"
    }
    // 更多内容……
  ]
}
```

在浏览器中打开后，页面效果如图 4-13 所示（使用 jsonview 插件做了 json 的代码格式化）。

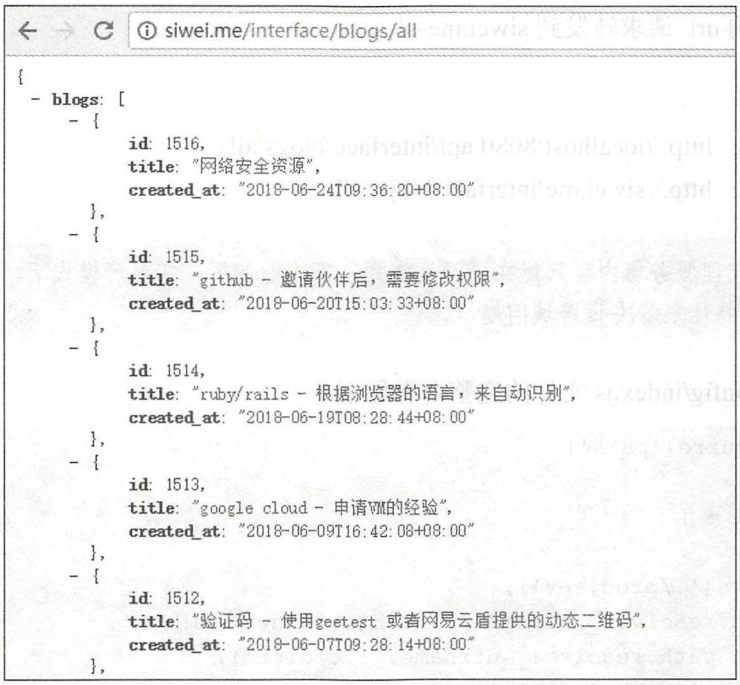


图 4-13　页面效果

4.6.3　设置 Vue.js 开发服务器的代理

正常来说，JavaScript 在浏览器中是无法发送跨域请求的，我们需要在 Vue.js 的“开发服务器”上做转发配置。

修改 config/index.js 文件，增加下列内容：

```
module.exports = {
  dev: {
    proxyTable: {
      '/api': {        // 1. 对所有以 "/api" 开头的 url 做处理
        target: 'http://siwei.me',   // 3. 转发到 siwei.me 上
        changeOrigin: true,
        pathRewrite: {
          '^/api': ''  // 2. 把 url 中的 "/api" 去掉
```

```
      }
    }
  },
}
```

上面的代码做了以下三件事。

（1）对所有以 "/api" 开头的 url 做处理。

（2）把 url 中的 "/api" 去掉。

（3）把新的 url 请求转发到 siwei.me 上。

例如：

- 原请求：http://localhost:8080/api/interface/blogs/all。
- 新请求：http://siwei.me/interface/blogs/all。

注 意

以上代理服务器内容只能在“开发模式”下才能使用。在生产模式下，只能靠服务器的 nginx 特性来解决 js 跨域问题。

修改后的 config/index.js 文件的完整内容如下：

```
var path = require('path')

module.exports = {
  build: {
    env: require('./prod.env'),
    index: path.resolve(__dirname, '../dist/index.html'),
    assetsRoot: path.resolve(__dirname, '../dist'),
    assetsSubDirectory: 'static',
    assetsPublicPath: '/',
    productionSourceMap: true,
    productionGzip: false,
    productionGzipExtensions: ['js', 'css'],
    bundleAnalyzerReport: process.env.npm_config_report
  },
  dev: {
    env: require('./dev.env'),
    port: 8080,
    autoOpenBrowser: true,
    assetsSubDirectory: 'static',
    assetsPublicPath: '/',

    proxyTable: {
      '/api': {
        target: 'http://siwei.me',
        changeOrigin: true,
```

```
      pathRewrite: {
        '^/api': ''
      }
    }
  },

  cssSourceMap: false
 }
}
```

重启服务器，可以看到转发设置已经生效。

```
$ npm run dev
...
[HPM] Proxy created: /api  ->  http://siwei.me
[HPM] Proxy rewrite rule created: "^/api" ~> ""
> Starting dev server...
...
```

4.6.4　打开页面，查看 http 请求

接下来，访问 http://localhost:8080/#/blogs/ 。

打开 chrome developer tools 就可以看到，"Network"中已经有请求发出去了，如图 4-14 所示的结果。

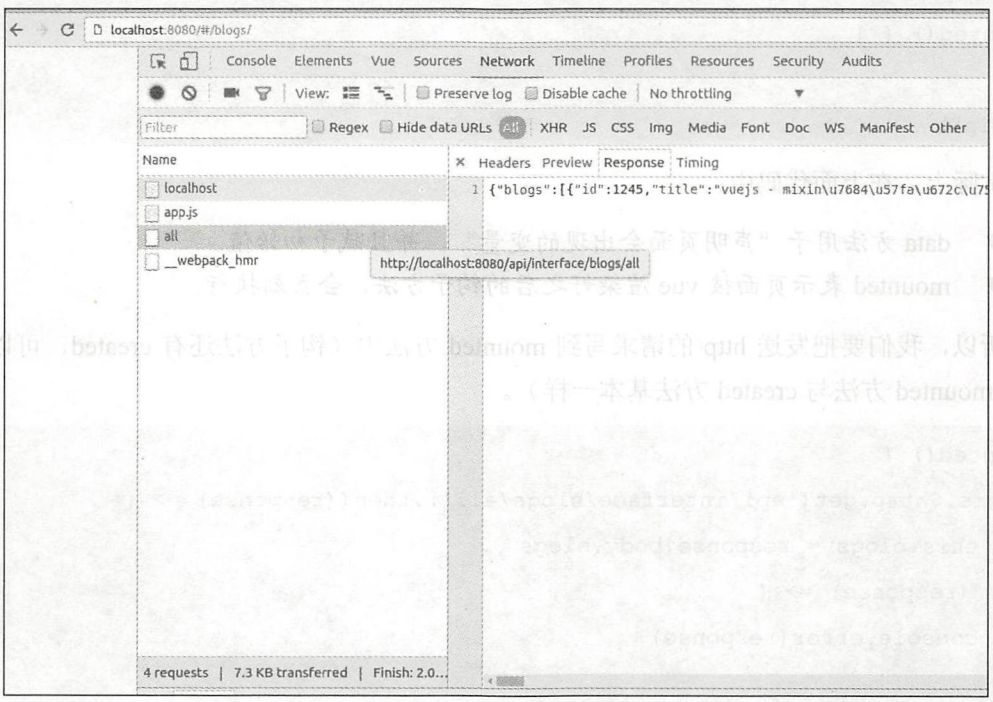


图 4-14　发出请求后

也可以直接在浏览器中输入要打开的链接，结果（该浏览器使用了 json view 插件）如图 4-15 所示。

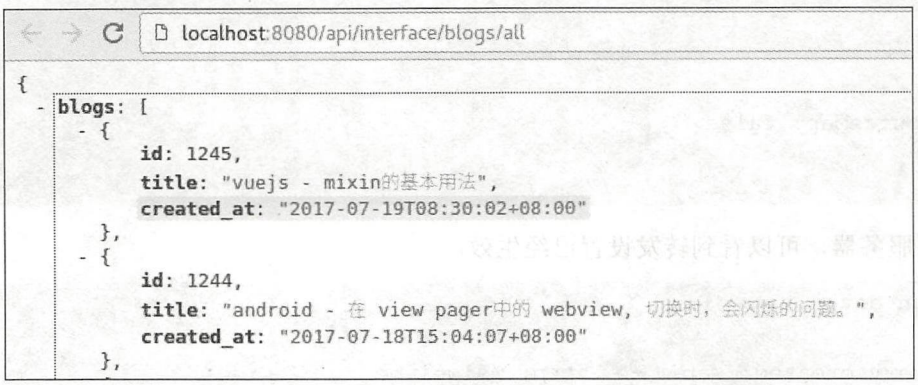


图 4-15　直接打开链接

4.6.5　把结果渲染到页面中

在 export 代码段中有以下两个部分。

```
<script>
export default {
  data () { },
  mounted() { }
}
</script>
```

实际上，在上面代码中：

- data 方法用于“声明页面会出现的变量”，并且赋予初始值。
- mounted 表示页面被 vue 渲染好之后的钩子方法，会立刻执行。

所以，我们要把发送 http 的请求写到 mounted 方法中（钩子方法还有 created，可以暂且认为 mounted 方法与 created 方法基本一样）。

```
  mounted() {
    this.$http.get('api/interface/blogs/all').then((response) => {
      this.blogs = response.body.blogs
    }, (response) => {
      console.error(response)
    });
  }
```

在上面的代码中：

- this.$http 中的 this 表示当前的 vue 组件（即 BookList.vue）；$http 表示所有以 $开头的变量都是 vue 的特殊变量，往往是 vue 框架自带。这里的$http 就是可以发起 http 请求的对象。
- $http.get 是一个方法，可以发起 get 请求，只有一个参数即目标 url。
- then()方法来自 promise，可以把异步请求写成普通的非异步形式。第一个参数是成功后的 callback，第二个参数是失败后的 callback。
- this.blogs = response.body.blogs 中，是把远程返回的结果（json）赋予到本地。因为 JavaScript 的语言特性能直接支持 JSON ，所以才可以这样写。

然后，我们通过这个代码进行渲染。

```
<tr v-for="blog in blogs">
  <td></td>
</tr>
```

在上面的代码中：

- v-for 是一个循环语法，可以把这个元素进行循环。注意，这个叫 directive 指令，需要与标签一起使用。
- blog in blogs: 前面的 blog 是一个临时变量，用于遍历使用；后面的 blogs 是 http 请求成功后，this.blogs = ...变量。同时，this.blogs 是声明于 data 钩子方法中。
- {{blog.title}} 用于显示每个 blog.title 的值。

4.6.6　如何发起 post 请求

与 get 类似，就是第二个参数是请求的 body。

在 vue 的配置文件中（如 Webpack 项目的 src/main.js 中）增加下面一句：

```
import VueResource from 'vue-resource';
Vue.use(VueResource);
....

//增加下面这句：
Vue.http.options.emulateJSON = true;
```

目的是为了能够让发出的 post 请求不会被浏览器转换为 option 请求。然后就可以按照下面的代码发送请求了。

```
this.$http.post('api/interface/blogs/all', {title: '', blog_body: ''})
    .then((response) => {
```

```
    ...
}, (response) => {
    ...
});
```

关于发送 http 请求的更多内容，可查看官方文档：https://github.com/pagekit/vue-resource

4.7 不同页面间的参数传递

在普通的 web 开发中，参数传递有以下几种形式。

- ur: /another_page?id=3。
- 表单：<form>...</form>。

而在 Vue.js 中，不会产生表单的提交（会引起页面的整体刷新），有以下两种。

- url: 同传统语言，参数体现在 url 中。
- Vue.js 内部的机制（无法在 url 中体现，可以认为是由 js 代码隐式实现的）。

我们用一个实际的例子说明：之前实现了“博客列表页”，接下来要实现“单击博客列表页中的某一行，就显示博客详情页”。

4.7.1 回顾：现有的接口

我们已经做好了一个接口：文章详情页。地址为：

```
/interface/blogs/show
```

该接口接收一个参数：id，使用 HTTP GET 请求进行访问。

下面是该接口的一个完整形式：http://siwei.me/interface/blogs/show?id=1244。

返回结果如下：

```
{
    "result":{
        "body":"<p>这个问题很常见，解决办法就是禁止硬件加速...</p>",
        "id":1244,
        "title":"android - 在 view pager 中的 webview，切换时，会闪烁的问题。"这个问题很常见，解决办法就是禁止硬件加速...</p>
    }
}
```

在浏览器中打开，结果如图 4-16 所示。

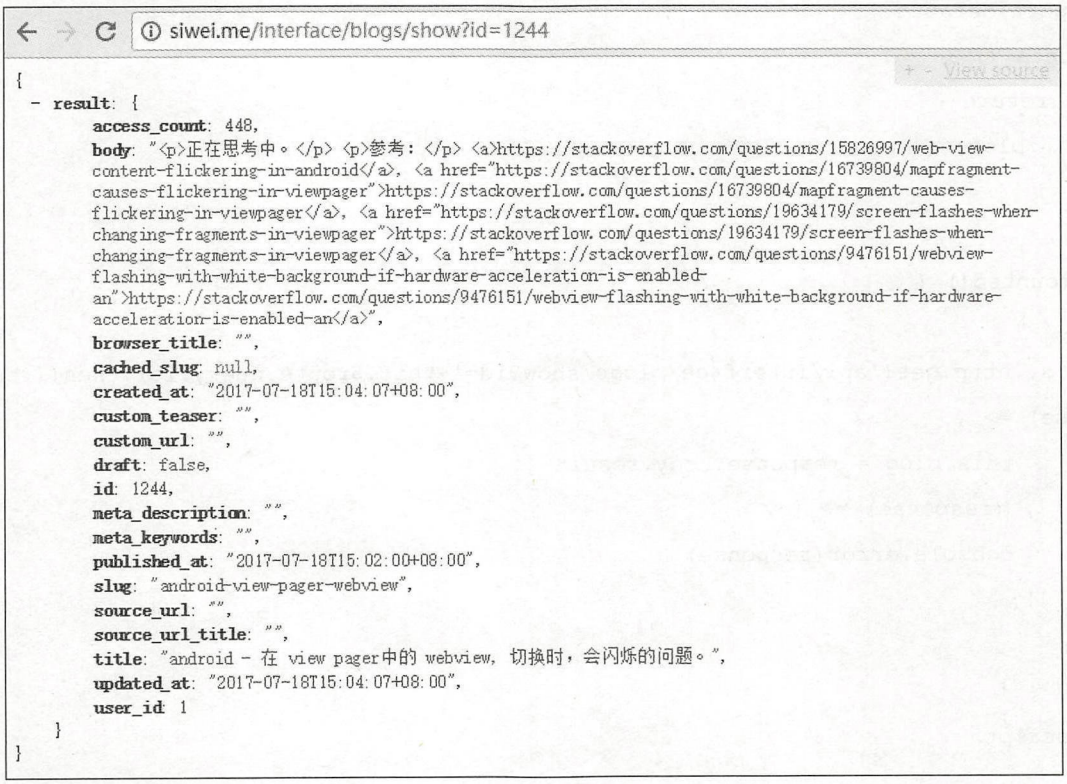


图 4-16　页面效果

4.7.2　显示博客详情页

我们需要新增 Vue 页面： src/components/Blog.vue，用于显示博客详情。

```
<template>
  <div >
    <div>
      <p> 标题：    </p>
      <p> 发布于：  </p>
      <div>

      </div>
    </div>
  </div>
</template>
```

```
<script>
export default {
  data () {
    return {
      blog: {}
    }
  },
  mounted() {

this.$http.get('api/interface/blogs/show?id='+this.$route.query.id).then((resp
onse) => {
      this.blog = response.body.result
    }, (response) => {
      console.error(response)
    });
  }
}
</script>

<style>
</style>
```

在上面的代码中：

- data(){ blog: {}}用于初始化 blog 页面用到的变量。
- {{blog.body}}、{{blog.title}}等用于显示 blog 相关的信息。
- mounted...中定义了发起 HTTP 的请求。
- this.$route.query.id 获取 url 中的 id 参数，如/my_url?id=333，'333'就是取到的结果。

4.7.3 新增路由

修改 src/router/index.js 文件。添加如下代码：

```
import Blog from '@/components/Blog'

export default new Router({
  routes: [
      // ...
    {
```


```
    path: '/blog',
    name: 'Blog',
    component: Blog
  }
 ]
} )
```

上面的代码就是让 Vue.js 可以对形如 http://localhost:8080/#/blog 的 url 进行处理。 对应的 vue 文件是 @/components/Blog.vue。

4.7.4　修改博客列表页的跳转方式 1：使用事件

我们需要修改博客列表页，增加跳转事件。修改 src/components/BlogList.vue，代码如下。

```
<template>
  ...
    <tr v-for="blog in blogs">
      <td @click='show_blog(blog.id)'></td>
    </tr>
  ...
</template>
<script>
export default {
  methods: {
    show_blog: function(blog_id) {
      this.$router.push({name: 'Blog', query: {id: blog_id}})
    }
  }
}
</script>
```

在上面的代码中：

- <td @click='show_blog(blog.id)'...</td>表示该<td>标签在被单击时会触发一个事件: show_blog，并且以当前正在遍历的 blog 对象的 id 作为参数。
- methods: {} 是比较核心的方法，vue 页面中用到的事件都要写在这里。
- show_blog: function...就是我们定义的方法。该方法可以通过@click="show_blog"调用。
- this.$router.push({name: 'Blog', params: {id: blog_id}})中，this.$router 是 vue 的内置对

象，表示路由。

- this.$router.push 表示让 vue 跳转，跳转到 name: Blog 对应的 vue 页面，参数是 id: blog_id 。

1. 演示结果

打开“博客列表页”，可以看到对应的文章，然后单击其中一篇文章的标题，就可以打开对应的文章详情页，如图 4-17 所示。

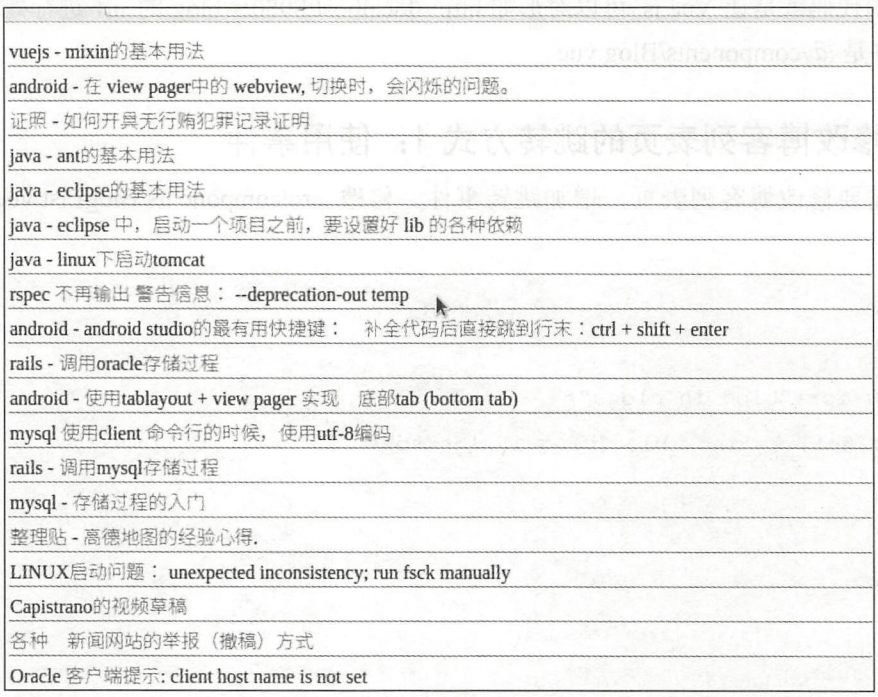


图 4-17 文章列表

2. 不经过 HTML 转义，直接打印结果

我们发现，HTML 的源代码在页面显示时被转义了，如图 4-18 所示。

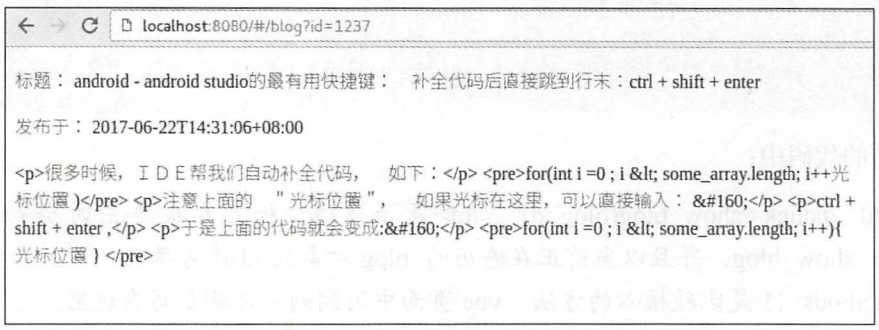


图 4-18 HTML 源代码被转义

所以，我们把它修改一下，即不转义。

```
<template>
  .....
    <div v-html='blog.body'>
    </div>
  .....
</template>
```

上面的 v-html 就表示不转义。页面效果如图 4-19 所示。

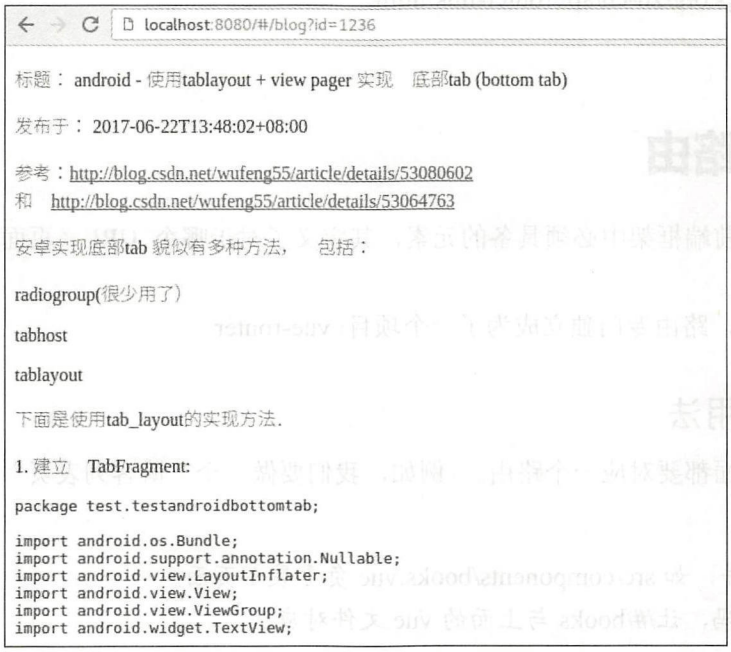


图 4-19　页面效果

4.7.5　修改博客列表页的跳转方式 2：使用 v-link

<router-link>比起<a href="...">要好一些。

因为无论是 HTML5 history 模式还是 hash 模式，其表现行为一致，所以当要切换路由模式，或者在 IE9 降级使用 hash 模式时，无须做任何变动。

在 HTML5 history 模式下，router-link 会拦截单击事件，让浏览器不再重新加载页面。

当用户在 HTML5 history 模式下使用 base 选项之后，所有的 to 属性都不需要写（基路径）了。

```
<td>
  <router-link :to="{name: 'Blog', query: {id: blog.id}}">

  </router-link>
```

```
</td>
```

然后就可以看到生成的 HTML 形如：

```
<a href="#/blog?id=1239" class="">
   1239
</a>
```

单击之后，有同样的跳转功能。感兴趣的读者，可以查看下面的链接：https://router.Vue.js.org/zh-cn/api/router-link.html。

4.8 路由

路由是所有前端框架中必须具备的元素，其定义了对于哪个 URL（页面）应该由哪个文件来处理。

在 Vue.js 中，路由专门独立成为了一个项目: vue-router。

4.8.1 基本用法

每个 vue 页面都要对应一个路由。 例如，我们要做一个“博客列表页”，就需要具备以下两个条件。

- vue 文件，如 src/components/books.vue 负责展示页面。
- 路由代码，让/#/books 与上面的 vue 文件对应。

下面的代码就是一个完整的路由文件。

```
import Vue from 'vue'
import Router from 'vue-router'

// 增加这一行，作用是引入 SayHi 这个 component
import SayHi from '@/components/SayHi'

Vue.use(Router)
export default new Router({
  routes: [

    // 增加下面几行，表示定义了  /#/say_hi 这个 url
    {
```

```
    path: '/say_hi',
    name: 'SayHi',
    component: SayHi
  },
 ]
})
```

写法是固定的，其中：

- path：定义了链接地址，如/#/say_hi。
- name 表示为这个路由加名字，方便以后引用，如 this.$router.push({name: 'SayHi'})。
- component 表示该路由由哪个 component 来处理。

4.8.2　跳转到某个路由时带上参数

有路由就会有参数，下面来看一下路由是如何处理参数的。

1. 对于普通的参数

例如：

```
routes: [
  {
    path: '/blog',
    name: 'Blog'
  },
]
```

在视图中，我们这样做：

```
<router-link :to="{name: 'Blog', query:{id: 3} }">User</router-link>
```

当用户单击这个代码生成的 html 页面时，就会触发跳转。

在<script/>中也可以这样做：

```
this.$router.push({name: 'Blog', query: {id: 3}})
```

都会跳转到/#/blog?id=3。

2. 对于在路由中声明的参数

例如：

```
routes: [
  {
```

```
    path: '/blog/:id',
    name: 'Blog'
  },
]
```

在视图中，我们这样做：

```
<router-link :to="{name: 'Blog', params: {id: 3} }">User</router-link>
```

在 script 中也可以这样做：

```
this.$router.push({name: 'Blog', params: {id: 3}})
```

都会跳转到/#/blog/3。

4.8.3 根据路由获取参数

在 Vue 的路由中，获取参数有两种方式：query 和 params。

1. 获取普通参数

对于/#/blogs?id=3 中的参数，可以这样获取：

```
this.$route.query.id  // 返回结果 3
```

2. 获取路由中定义好的参数

对于/#/blogs/3 这样的参数，可以对应的路由应该是：

```
routes: [
  {
    path: '/blogs/:id',  // 注意这里的 :id
    ...
  },
]
```

这个 named path 就可以通过下面的代码来获取 id。

```
this.$route.params.id  // 返回结果 3
```

4.9 使用样式

样式用起来特别简单，直接写到<style>段落里面即可。代码如下：

```
<template>
  <div class='hi'>
    Hi Vue!
  </div>
</template>

<script>
export default {
  data () {
    return { }
  }
}
</script>

<style>
.hi {
  color: red;
  font-size: 20px;
}
</style>
```

使用浏览器打开上述代码，就可以看到一个红色的，字体大小为 20px 的 Hi Vue!，如图 4-20 所示。

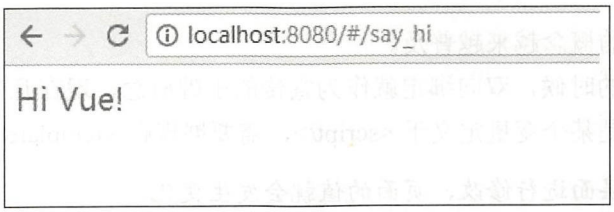


图 4-20　页面效果

1. 使用全局

```
<style >
td {
  border-bottom: 1px solid grey;
}
</style>
```

2. 使用局部的 CSS

```
<style scoped>
td {
  border-bottom: 1px solid grey;
}
</style>
```

这段 CSS 只对当前的 component 适用。

也就是说，当我们有两个不同的页面（page1 和 page2）时，如果两个页面中都定义了某个样式（如上面的 td），是不会互相冲突的。

因为 Vue.js 会这样解析：

```
page1 的 DOM:
<div data-v-7cfd41e ... ></div>

page2 的 DOM:
<div data-v-3389dfw ... ></div>
```

而我们使用的"scoped style"就可以存在于不同的页面（component）上了。

4.10 双向绑定

现在，双向绑定的概念越来越普及。

在 Angular 出现的时候，双向绑定就作为宣传的王牌概念，现在几乎每个 js 前端框架就有该功能。它的概念是某个变量定义于 <script/>，需要展现在 <template/>中的话：

- 如果在代码层面进行修改，页面的值就会发生变化。
- 如果在页面进行修改（如在 input 标签中），代码的值就会发生变化。

在我们的项目中，增加一个 vue 页面：src/components/TwoWayBinding.vue。

```
<template>
  <div>
    <!-- 显示  this.my_value 变量 -->
    <p>页面上的值:  </p>

    <p> 通过视图层，修改 my_value: </p>
```

```
    <input v-model="my_value" style='width: 400px'/>

    <hr/>
    <input type='button' @click="change_my_value_by_code()" value='通过控制代码修改my_value'/>
    <hr/>
    <input type='button' @click="show_my_value()" value='显示代码中的my_value'/>
  </div>
</template>

<script>
export default {
  data () {
    return {
      my_value: '默认值',
    }
  },
  methods: {
    show_my_value: function(){
      alert('my_value: ' + this.my_value);
    },
    change_my_value_by_code: function(){
      this.my_value += ", 在代码中做修改, 666."
    }
  }
}
</script>
```

在上面的代码中显示定义了一个变量：my_value，该变量可以在 <script/>中访问和修改，也可以在<template/>中访问和修改。

- 在代码（<script/>）中访问，就是 this.my_value。
- 在视图（<template/>）中访问，就是<input v-model=my_value />。

这个就是双向绑定的方法。

接下来，修改路由文件: src/router/index.js。

```
import TwoWayBinding from '@/components/TwoWayBinding'
```

```
export default new Router({
  routes: [
    {
      path: '/two_way_binding',
      name: 'TwoWayBinding',
      component: TwoWayBinding
    }
  ]
})
```

然后可以使用浏览器访问路径：http://localhost:8080/#/two_way_binding。

效果 1：通过页面修改 js 代码的值，可以看到，一旦代码中的 my_value 发生改变，视图中的 my_value 就发生变化，如图 4-21 所示。

图 4-21　效果 1

效果 2：通过代码层面的改动影响页面的值，如图 4-22 所示。

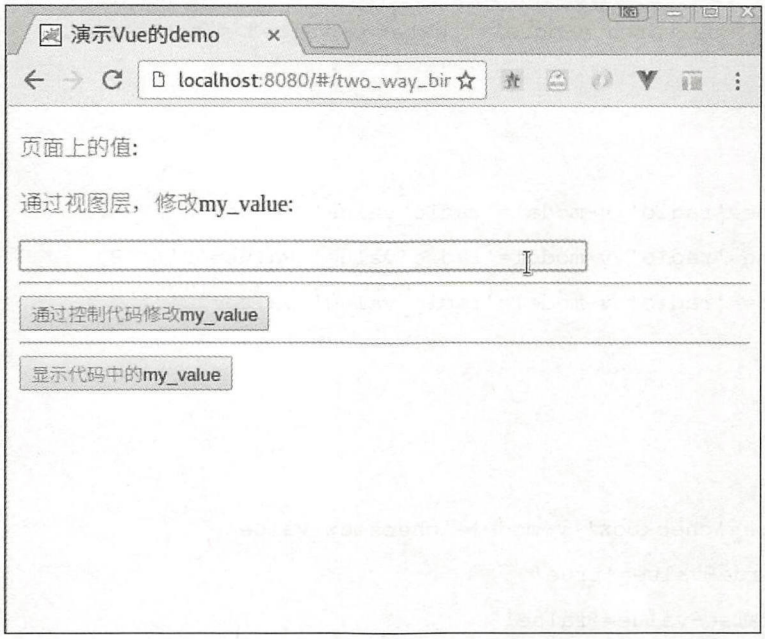


图 4-22　效果 2

这个特性是 Vue.js 自带的，我们不需要刻意学习，只需要知道它可以达到这个目就可以了。读者以后会发现，这种思想和现象在 Vue.js 等前端框架中特别常用。

4.11 表单项目的绑定

所有的表单项，无论是<input/>还是<textarea/>，基本上都需要使用 v-model 来绑定。

1. 表单项 input、textarea、select 等

使用 v-model 来绑定输入项。

```
<input v-model="my_value" style='width: 400px'/>
```

可以在代码中获取到 this.my_value 的值。

3. 表单项的完整例子

```
<template>
  <div>

    input:  <input type='text' v-model="input_value"/>,
    输入的值:
    <hr/>
```

```
    text area:  <textarea v-model="textarea_value"></textarea>,
    输入的值：
    <hr/>
    radio:
    <input type='radio' v-model='radio_value' value='A'/> A,
    <input type='radio' v-model='radio_value' value='B'/> B,
    <input type='radio' v-model='radio_value' value='C'/> C,
    输入的值：

    <hr/>
    checkbox:
    <input type='checkbox' v-model='checkbox_value'
      v-bind:true-value='true'
      v-bind:false-value='false'
      /> ,
    输入的值：

    <hr/>
    select:
    <select v-model='select_value'>
      <option v-for="e in options" v-bind:value="e.value">

      </option>
    </select>
    输入的值：

  </div>
</template>

<script>
export default {
  data () {
    return {
      input_value: '',
      textarea_value: '',
      radio_value: '',
```

```
    checkbox_value: '',
    select_value: 'C',
    options: [
      {
        text: '红烧肉', value: 'A'
      },
      {
        text: '囊包肉', value: 'B'
      },
      {
        text: '水煮鱼', value: 'C'
      }
    ]
  }
 },
 methods: {
 }
}
</script>
```

对于 select 的 option，使用 v-bind:value 来绑定 option 的值，效果如图 4-23 所示。

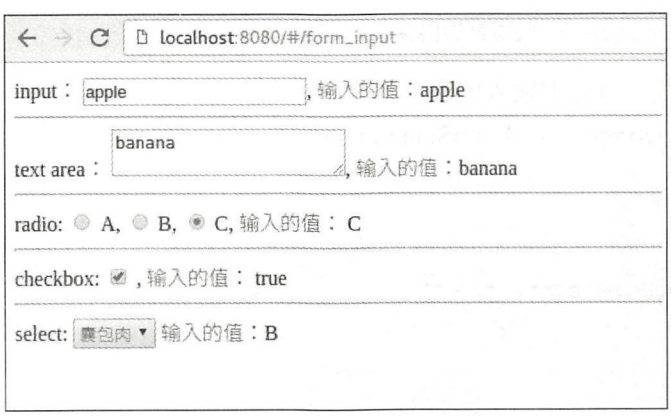


图 4-23　页面效果

4. Modifiers （后缀词）

（1）.lazy

在用户对某个文本框做输入的时候，文本框中的值不会随着用户按下的每一个键立刻发生变化，而是等焦点彻底离开文本框后（触发 blur()事件后）触发视图中值的变化。

使用方式如下：

```
<input type='text' v-model.lazy="input_value"/>
```

这个可以用在某些需要等待用户输入完字符串再需要给出反应的情况，如“搜索”。

（2）.number

强制要求输入数字。使用方式如下：

```
<input type='text' v-model.lazy="input_value" type="number"/>
```

（3）.trim

强制对输入的值去掉前后的空格。使用方式如下：

```
<input type='text' v-model.trim="input_value" />
```

4.12 表单的提交

在任何 Single Page App 中，js 代码都不会产生一个传统意义的 form 表单提交（这会引起整个页面的刷新），一般用事件来实现（桌面开发思维）。

例如，在远程有一个接口，可以接受别人的留言：

- URL：http://siwei.me/interface/blogs/add_comment。
- 参数：content（留言的内容）。
- 请求方式：POST。
- 返回结果的例子如下。

```
{"result":"ok","content":"(留言的内容)"}
```

例如，下面的代码就是把输入的表单提交到后台。

新增加一个/src/components/FormSubmit.vue 文件，内容如下：

```
<template>
  <div>
    <textarea v-model='content'>
    </textarea>
    <br/>
    <input type='button' @click='submit' value='留言'/>
  </div>
</template>
<script>

export default {
```

```
  data () {
    return {
      content: ''
    }
  },
  methods: {
    submit: function(){
      this.$http.post('/api/interface/blogs/add_comment',
        {
          content: this.content
        }
      )
      .then((response) => {
          alert("提交成功!, 刚才提交的内容是: " + response.body.content)
        },
        (response) => {
          alert("出错了")
        }
      )
    }
  }
}
</script>
```

从上面的代码中可以看到：

（1）下面的代码是待输入的表单项。

```
<textarea v-model='content'>
</textarea>
```

（2）下面的代码则是按钮被单击后触发提交表单的函数 submit。

```
<input type='button' @click='submit' value='留言'/>
```

（3）下面的代码定义了提交表单的函数 submit。

```
  submit: function(){
    this.$http.post('/api/interface/blogs/add_comment',
      {
```

```
        content: this.content
      }
    )
    .then((response) => {
      alert("提交成功!, 刚才提交的内容是: " + response.body.content)
    },
    (response) => {
      alert("出错了")
    }
    )
  }
```

- this.$http.post 表示发起的 HTTP 类型是 post。
- post 函数的第一个参数是 url，第二个参数是 json，{ content: this.content}代表了要提交的数据。
- then 函数的处理同 HTTP get 请求。

接下来，修改路由 src/router/index.js 文件，增加内容如下：

```
import FormSubmit from '@/components/FormSubmit'

export default new Router({
  routes: [
    {
      path: '/form_submit',
      name: 'FormSubmit',
      component: FormSubmit
    }
  ]
} )
```

访问 url: http://localhost:8080/form_submit，输入一段字符串，如图 4-24 所示。

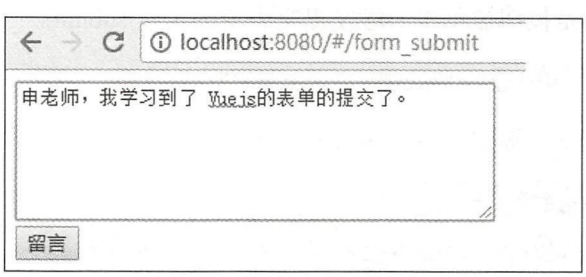


图 4-24　输入一段字符串

单击提交按钮就可以看到，内容已经提交，并且得到了返回的 response，触发了 alert，如图 4-25 所示。

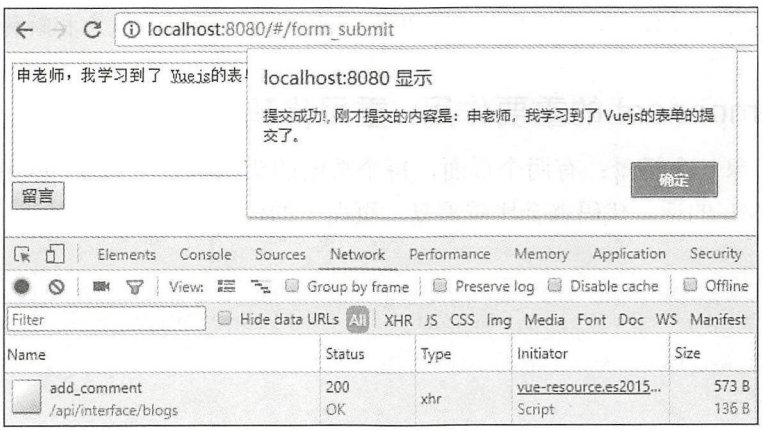


图 4-25　提交成功

查看一下返回的 JSON：

```
{"result":"ok","content":"\u7533\u8001\u5e08\uff0c\u6211\u5b66\u4e60\u5230\u4e
86 Vue.js\u7684\u8868\u5355\u7684\u63d0\u4ea4\u4e86\u3002"}
```

至此，完成了一个完整的输入表单、提交表单的过程。

4.13 Component 组件

组件是 Vue.js 中最重要的部分之一，学好组件知识需要一定的时间投入。在 Webpack 项目中，每一个页面文件（.vue）都可以认为是一个组件。

在 Vue.js 1.x 中，组件与视图是分别放到不同的文件夹下面的，在 Vue.js 2.8 以后，所有的视图文件都保存在 components 目录下。

可见，组件的概念已经越来越普及了。

注　意

这里的内容，与官方文档中的“原始组件”不一样。本章的内容，仅使用于 Webpack 项目中的组件，官方网站中对应的文档是单文件组件。

4.13.1　如何查看文档

先快速查看官方文档中关于“原始组件”的页面：https://cn.Vue.js.org/v2/guide/single-file-components.html，对所有的概念有所了解，因为这个“原始组件”的开发环境与 webpack 下项目的开发环境不太一样，所以很多以 webpack 作为入门的读者（如本书读者）会感到迷

茫。再看“单文件组件”：https://cn.Vue.js.org/v2/guide/single-file-components.html，就可以对webpack 项目下的组件有清晰地认识。

遇到问题之后，再看“原始组件”，这里包括很多 API 级别的概念和解释。

4.13.2 Component 的重要作用：重用代码

我们可以想象一个场景：有两个页面，每个页面的头部都有一张 Logo 图片。如果每次都写成原始 HTML 的话，代码就会比较重复。页面 1 的代码如下：

```
  <div class='logo'>
    <img src='http://siweitech.b0.upaiyun.com//image/569/upsz-
fyfkzhs9232258.jpg'>
  </div>
  页面 1 的其他代码
```

页面 2 的代码如下：

```
  <div class='logo'>
    <img src='http://siweitech.b0.upaiyun.com//image/569/upsz-
fyfkzhs9232258.jpg'>
  </div>
  页面 2 的其他代码
```

因此，我们应该把这段代码抽取出来成为一个新的组件(component)。

4.13.3 组件的创建

新建一个 src/components/Logo.vue 文件。

```
<template>
  <div class='logo'>
    <img src='http://siweitech.b0.upaiyun.com//image/570/siwei.me_header.png'/>
  </div>
</template>
```

该文件中定义了一个比较简单的 component。然后修改对应的页面：

```
<template>
  <div >
    <my-logo>
    </my-logo>
      ...
```


```
</template>

<script>
import MyLogo from '@/components/Logo'

export default {
    ...
  components: {
    MyLogo
  }
}
```

上面代码中的 components: { MyLogo }必须是这个写法，等同于：

```
components: {
  MyLogo: MyLogo   // 前面的 MyLogo 是 template 中的名字
                   // 后面的 MyLogo 是 import 进来的代码
}
```

虽然上面代码中定义的组件名字为 MyLogo，但是在<template/>中使用时需要写为<my-logo></my-logo>。

保存代码并刷新一次，发现两个页面都发生了变化，如图 4-26 所示。

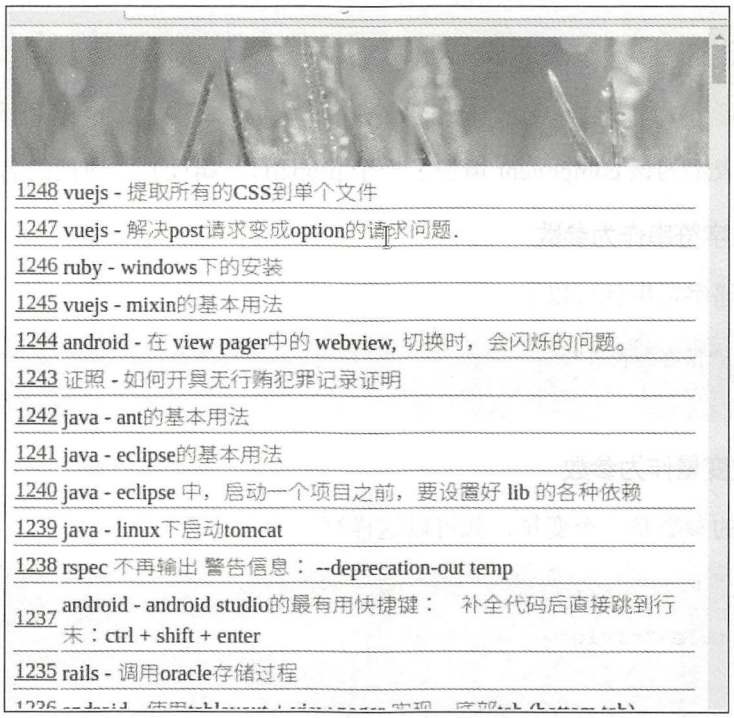


图 4-26　两个页面都发生了变化

4.13.4 向组件中传递参数

如果希望两个页面中都有一个 title，内容却不同，该怎么办呢？这时就需要向 Component 传递参数了。

声明组件时，需要修改 src/components/Logo.vue 文件。

```
<template>
  <div class='logo'>
    <h1></h1>
        ...
  </div>
</template>

<script>
export default {
  props: ['title']    // 加上这个声明.
}
</script>
```

可以看到，在上面的代码中增加了以下几行代码。

```
export default {
  props: ['title']
}
```

上面的代码表示为该 component 增加了一个 property（属性），属性的名字为 title。

1. 组件接收字符串作为参数

在调用时传递字符串就可以了。

```
<my-logo title="博客列表页">
</my-logo>
```

2. 组件接收变量作为参数

如果要传递的参数是一个变量，就可以这样写：

```
<template>
    <my-logo :title="title">
    </my-logo>
    <input type='button' @click='change_title' value='单击修改标题'/><br/>
</template>
```


```
<script>
export default {
  data: function() {
    return {
      title: '博客列表页',
    }
  },
  methods: {
    change_title: function(){
      this.title = '好多文章(标题被代码修改过了)'
    }
  },
</script>
```

如图 4-27 所示。

图 4-27　标题已被修改

4.13.5 脱离 Webpack，在原生 Vue.js 中创建 component

在原生 Vue.js 中创建 component 的过程非常简单。代码如下：

```
<html>
<head>
  <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
</head>
<body>
  <div id='app'>
    <study-process></study-process>
  </div>
  <script>
    Vue.component('study-process', {
      data: function () {
        return {
          count: 0
        }
      },
      template: '<button v-on:click="count++">我学习到了第  章.</button>'
    })
    var app = new Vue({
      el: '#app',
      data: {
      }
    })
  </script>
</body>
</html>
```

该代码首先声明了一个 component：

```
    Vue.component('study-process', {
      data: function () {
        return {
          count: 0
        }
      },
```

```
    template: '<button v-on:click="count++">我学习到了第  章.</button>'
})
```

可以看出，该 component 定义了一个 data 代码段，其中有一个 count 变量，然后定义一个 template 段落即可。

第 5 章

运维和发布Vue.js项目

一般传统公司特别喜欢一个角色用一个人，如在生产流水线上，一个工人只负责拧螺丝，另一个工人只负责喷漆。很多软件公司也是这样，有人专门负责编写代码，有人专门负责运维。但是这样的弊端是：出了问题，程序员接触不到服务器，不知道问题出在哪里，运维可以接触服务器，却看不懂代码，也没有办法解决问题。所以传统公司往往怕出问题，因为解决起来特别慢。

现在，越来越多的人意识到，让程序员懂得运维知识特别重要。最好的运维就是程序员自己。在 2011 年，国外就开始流行一个词汇：DevOps（Developer + Operations），也叫作敏捷运维，就是对既懂编程又懂运维的人的称呼。

我们要有追求，做一个会运维的编程高手，做一个 DevOps。

5.1 打包和部署

我们平时都是在本地做开发，每次打开的都是 http://localhost:8080/#/，而在真实的项目中，需要把项目部署到某个地方，对项目进行打包和编译。

另外，在产品正式上线之前，也要在测试服务器上进行发布，这样才能发现一些平时在 localhost 上看不到的问题。

5.1.1 打包

直接使用下面的命令就可以把 vue 项目打包。

```
$ npm run build
```

该命令的运行过程如下：

```
siwei@siwei-linux:/workspace/test_vue_0613$ npm run build

> test_vue_0613@1.0.0 build /workspace/test_vue_0613
```

```
> node build/build.js

∴ building for production...
Starting to optimize CSS...
Processing static/css/app.32ddfe6eea5926f8e3c760d764fef3fa.css...
Processed static/css/app.32ddfe6eea5926f8e3c760d764fef3fa.css, before: 142,
after: 74, ratio: 52.11%
Hash: f89cd58bdaf8a153e13e
Version: webpack 2.6.1
Time: 18658ms
                                        Asset       Size  Chunks
Chunk Names
               static/js/app.d8b9f437c302a7070fe7.js     9.1 kB     0  [emitted]
app
            static/js/vendor.33c767135f1684f458a7.js     122 kB       1
[emitted]  vendor
           static/js/manifest.75e2ba037e0bc6934514.js    1.51 kB       2
[emitted]  manifest
   static/css/app.32ddfe6eea5926f8e3c760d764fef3fa.css   74 bytes       0
[emitted]  app
            static/js/app.d8b9f437c302a7070fe7.js.map    63.5 kB       0
[emitted]  app
static/css/app.32ddfe6eea5926f8e3c760d764fef3fa.css.map  623 bytes       0
[emitted]  app
         static/js/vendor.33c767135f1684f458a7.js.map     950 kB       1
[emitted]  vendor
       static/js/manifest.75e2ba037e0bc6934514.js.map    14.6 kB       2
[emitted]  manifest
                                    index.html  522 bytes          [emitted]

  Build complete.

  Tip: built files are meant to be served over an HTTP server.
  Opening index.html over file:// won't work.
```

可以看到：

（1）整个过程耗时 18.658s。

（2）使用的 Webpack 版本是 2.6.1。

（3）对 CSS 文件进行了优化（优化的比率是 52.11%）。

（5）所有的.vue 文件都被打包编译成了下面的文件。

```
$ find ./dist
.
./static
./static/css
./static/css/app.32ddfe6eea5926f8e3c760d764fef3fa.css
./static/css/app.32ddfe6eea5926f8e3c760d764fef3fa.css.map
./static/js
./static/js/vendor.33c767135f1684f458a7.js.map
./static/js/app.d8b9f437c302a7070fe7.js.map
./static/js/manifest.75e2ba037e0bc6934514.js
./static/js/manifest.75e2ba037e0bc6934514.js.map
./static/js/app.d8b9f437c302a7070fe7.js
./static/js/vendor.33c767135f1684f458a7.js
./index.html
```

其中包括 js、css、map 和 index.html。

我们需要将其放到 HTTP 服务器上，如 nginx 、apache。

5.1.2 部署

1. 上传代码到远程服务器

使用 scp 或 ftp 的方式，可以把代码上传到服务器。假设服务器是 linux：

- 路径是/opt/app/test_vue;
- 服务器 IP 是 123.255.255.33;
- 服务器 ssh 端口是 6666;
- 服务器用户名是 root。

```
$ scp -P 6666 -r dist root@123.255.255.33:/opt/app
index.html                                    100%  528     0.5KB/s   00:00
app.32ddfe6eea5926f8e3c760d764fef3fa.css      100%   74     0.1KB/s   00:00
app.32ddfe6eea5926f8e3c760d764fef3fa.css.map  100%  623     0.6KB/s   00:00
```

```
vendor.33c767135f1684f458a7.js.map            100%  927KB 927.3KB/s   00:00
app.d8b9f437c302a7070fe7.js.map               100%   63KB  62.6KB/s   00:00
manifest.75e2ba037e0bc6934514.js              100% 1511     1.5KB/s   00:00
manifest.75e2ba037e0bc6934514.js.map          100%   14KB  14.3KB/s   00:00
app.d8b9f437c302a7070fe7.js                   100% 9323     9.1KB/s   00:00
vendor.33c767135f1684f458a7.js                100%  119KB 118.7KB/s   00:00
```

这样就把本地的 dist 目录，上传到了远程的/opt/app 目录上。

2. 配置远程服务器

（1）登录远程服务器。

```
$ ssh root@123.255.255.23 -p 6666
(输入密码，  回车)

Welcome to Ubuntu 14.04.4 LTS (GNU/Linux 3.13.0-86-generic x86_64)

root@my_server:~#
```

（2）把刚才上传的文件夹重命名为 vue_demo。

```
# mv /opt/app/dist /opt/app/vue_demo
```

（3）配置 nginx，使域名：vue_demo.siwei.me 指向该位置。把下面的代码加入到 nginx 的配置文件中（/etc/nginx/nginx.conf）：

```
server {
        listen       80;
        server_name  vue_demo.siwei.me;
        client_max_body_size      500m;
        charset utf-8;
        root /opt/app/vue_demo;
}
```

（4）重启 nginx 之前测试一下刚才加入的代码是否有问题。

```
# nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
```

（5）可以看到代码没问题，然后重启 nginx ：

```
# nginx -s stop
# nginx
```

3. 修改域名配置

nginx 运行之后，接下来就是配置域名，否则无法访问。我们需要增加一个二级域名：vue_demo.siwei.me。

不同的域名服务商提供的操作界面也不一样，根本思路就是增加 A 地址。我的域名服务商是 dnspod，登录后，可以看到我的域名列表，如图 5-1 所示的域名为 siwei.me 的记录管理页面。

图 5-1　域名为 siwei.me 的记录管理页面

单击“添加记录”按钮，就可以看到该域名的编辑页面，如图 5-2 所示的增加 vue_demo 二级域名的 A 记录。

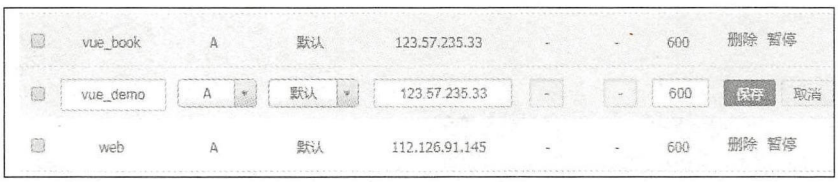


图 5-2　增加 vue_demo 二级域名的 A 记录

输入对应的二级域名（vue_demo），选择记录类型为 A，记录值为 siwei.me 的主机 IP 地址，然后单击“保存”按钮。再回到命令行，输入 ping 命令。

```
$ ping vue_demo.siwei.me
PING vue_demo.siwei.me (123.57.235.33) 56(84) bytes of data.
64 bytes from 123.57.235.33: icmp_seq=1 ttl=54 time=5.79 ms
64 bytes from 123.57.235.33: icmp_seq=2 ttl=54 time=6.38 ms
64 bytes from 123.57.235.33: icmp_seq=3 ttl=54 time=9.25 ms
```

上面的结果说明二级域名 vue_demo.siwei.me 已经可以正常指向到服务器 123.57.235.33（这个就是我们本次 demo 的后端服务器的 IP 地址）了。

4. 完成部署

打开浏览器，访问 http://vue_demo.siwei.me 就可以看到结果了，如图 5-3 所示。

图 5-3　项目可以通过 vue_demo.siwei.me 访问

5.2 解决域名问题与跨域问题

我们在部署之后会发现 Vue.js 遇到 js 的经典问题：远程服务器地址不对，或者跨域问题。还是以本书中的“4.6 发送 http 请求”（显示博客的列表）为例。

真正的后台接口是 http://siwei.me/interface/blogs/all，如图 5-4 所示。

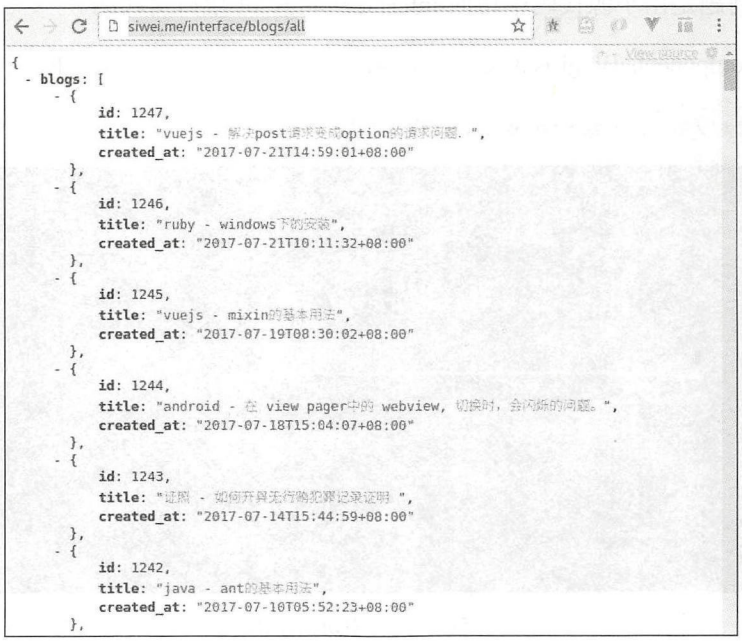


图 5-4　后台接口

5.2.1 域名 404 问题

（1）使用浏览器打开 http://vue_demo.siwei.me/#/blogs 页面，提示页面出错，如图 5-5 所示。

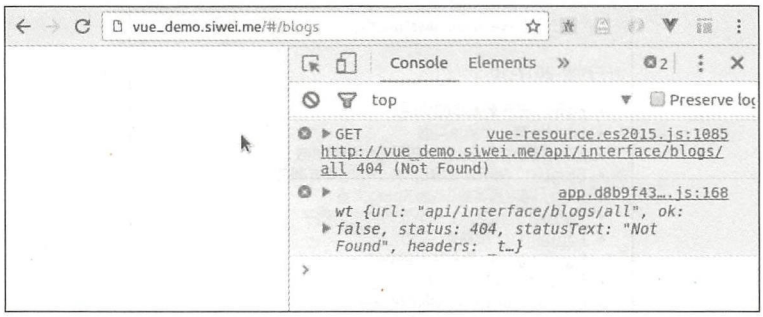


图 5-5 页面出错

（2）可以看到出错的原因是 404，打开 http://vue_demo.siwei.me/api/interface/blogs/all 页面，如图 5-6 所示接口地址返回 404 错误。

图 5-6 接口地址返回 404 错误

（3）这个问题是由于源代码中访问/interface/blogs/all 接口引起的。

在文件 src/components/BlogList.vue 中的第 41 行，定义了远程访问的 URL，如图 5-7 所示。

```
this.$http.get('/api/interface/blogs/all')...
```

```
31  methods: {
32    show_blog: function(blog_id) {
33      console.info("blog_id:" + blog_id)
34      this.$router.push({name: 'Blog', query: {id: blog_id}})
35    },
36    change_title: function(){
37      this.title = '好多文章啊(标题被代码修改过了)'
38    }
39  },
40  mounted() {
41    this.$http.get('/api/interface/blogs/all').then((response) => {
42      console.info(response.body)
43      this.blogs = response.body.blogs
44    }, (response) => {
45      console.error(response)
46    });
47  },
48  components: {
49    MyLogo
50  }
```

图 5-7 代码中访问 URL

这是因为在开发时，Vue.js 会通过 $npm run dev 命令来运行“本地开发服务器”。这个服务器中有一个代理，可以把所有的以 '/api' 开头的请求，如：

```
localhost:8080/api/interface/blogs/all
```

转发到：

```
siwei.me/interface/blogs/all
```

“本地开发服务器”的配置如下：

```
proxyTable: {
  '/api': {
    target: 'http://siwei.me',
    changeOrigin: true,
    pathRewrite: {
      '^/api': ''
    }
  }
},
```

所以，在开发环境下一切正常。但是在生产环境中发起请求时，就不存在“代理服务器”“开发服务器”（dev server）了，因此会出错。

5.2.2　跨域问题

跨域是 js 的经典问题。比如，有的读者在解决上面的问题时，会问：我们直接把图 5-7 中 41 行的：

```
this.$http.get('/api/interface/blogs/all')
```

改成：

```
this.$http.get('http://siwei.me/interface/blogs/all')
```

不就可以了吗？答案是否定的。

动手试一下我们就会发现，如果 vue_demo.siwei.me 直接访问 siwei.me 域名下的资源，就会报错。因为它们是两个不同的域名。

代码如：

```
this.$http.get('http://siwei.me/api/interface/blogs/all')...
```

就会得到报错：

```
XMLHttpRequest cannot load http://siwei.me/api/interface/blogs/all.
```

```
No 'Access-Control-Allow-Origin' header is present on the requested resource.
Origin 'http://vue_demo.siwei.me' is therefore not allowed access.
```

如图 5-8 所示的跨域问题的报错信息。

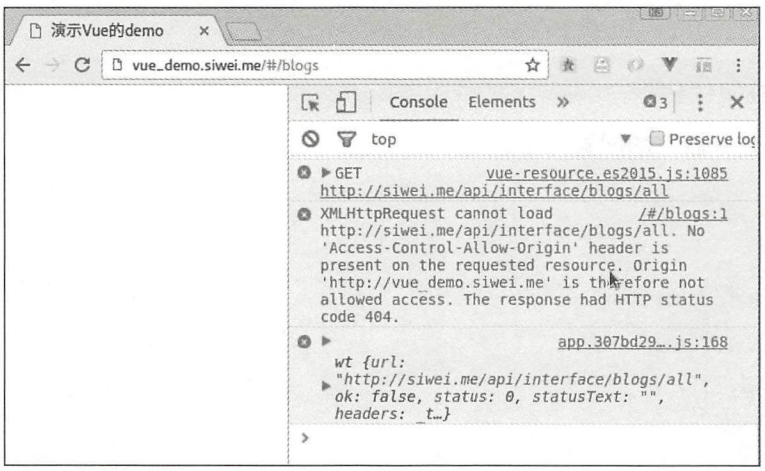


图 5-8　跨域问题的报错信息

5.2.3　解决域名问题和跨域问题

其实，上面提到的两个问题的根源都是一个，也就是说，解决办法是相同的。

（1）在代码端，处理方式不变，访问/api+原接口 url。（无变化）

```
this.$http.get('/api/interface/blogs/all')...
```

（2）在开发时继续保持 Vue.js 的代理存在。配置代码如下：（无变化）

```
proxyTable: {
  '/api': {
    target: 'http://siwei.me',
    changeOrigin: true,
    pathRewrite: {
      '^/api': ''
    }
  }
},
```

（3）在 nginx 的配置文件中加入代理（详细说明参见代码中的注释，这个是新增的）。

```
  server {
    listen       80;
    server_name  vue_demo.siwei.me;
```

```
    client_max_body_size       500m;
    charset utf-8;
    root /opt/app/vue_demo;

    # 第一步,把所有的 mysite.com/api/interface  转换成   mysite.com/interface
    location /api {
      rewrite    ^(.*)\/api(.*)$    $1$2;
    }

    # 第二步,把所有的 mysite.com/interface 的请求转发到 siwei.me/interface
    location /interface {
      proxy_pass          http://siwei.me;
    }
  }
```

就可以了

也就是说，上面的配置把：

```
http://vue_demo.siwei.me/api/interface/blogs/all
```

在服务器端的 nginx 中做了变换，相当于访问了：

```
http://siwei.me/interface/blogs/all
```

重启 nginx，就会发现已经生效了。恢复正常的文章列表页如图 5-9 所示。

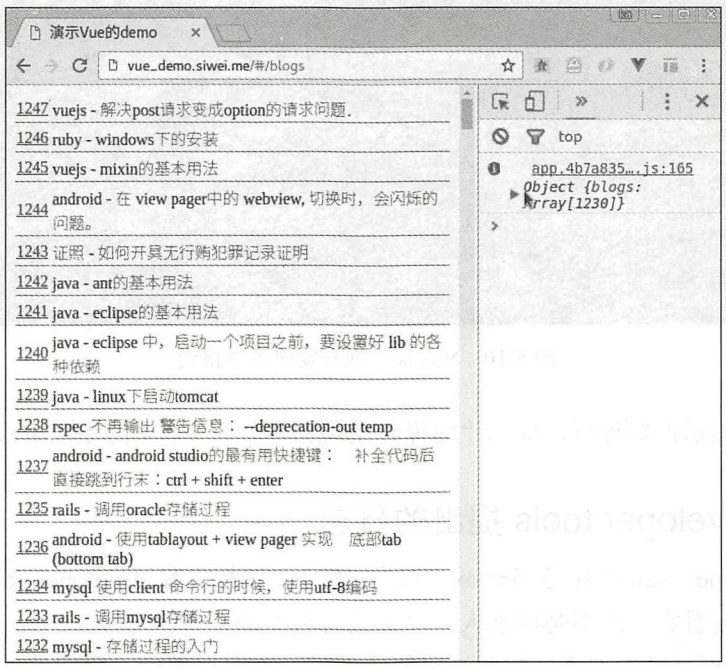


图 5-9　恢复正常的文章列表页

5.3 如何 Debug

浏览器环境下的 JavaScript，实际上有两个天生的缺陷。

- 不严谨。不同浏览器的 js 实现上会略有不同，这个问题在 Android、iOS 上也是一样的。
- 不是严格意义上的计算编程语言，有语法漏洞，如 "=="。

因此，我们要想驾驭好 JS 语言，就要知道如何有效的 Debug。

5.3.1 时刻留意本地开发服务器

开发时的命令如下：

```
$ npm run dev
```

会开启“本地开发服务器”，要时刻留意该服务器的后台输出。有时候我们把代码编写错了，导致 Vue.js 无法编译，浏览器页面就会一片空白，并且没有任何出错提示。

其实浏览器页面的一片空白，是由于 Vue.js 的代码无法运行导致的。此时服务器会报错，如图 5-10 所示的 Vue.js 本地开发服务器报错。

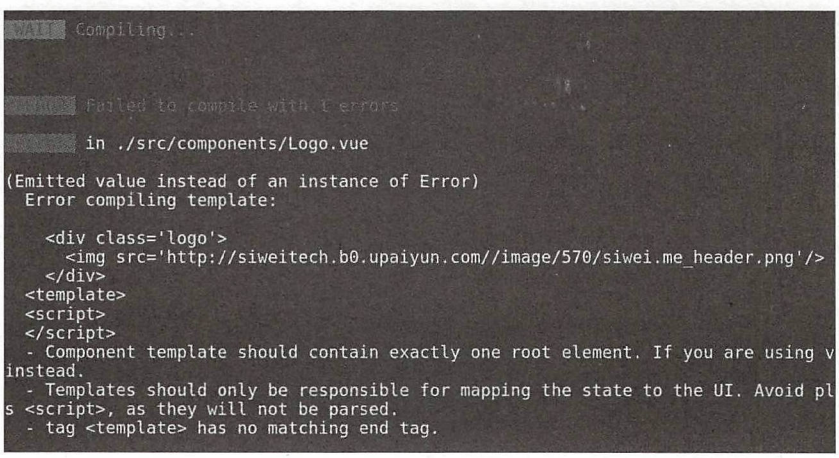


图 5-10　Vue.js 本地开发服务器报错

上面的错误提示很好理解，提示“编译时出现错误”，并给出了错误的详细位置。

5.3.2 看 developer tools 提出的日志

无论是 chrome、safari 还是 firefox，以及 IE 7+，都带有 developer tools。任何时候页面空白了，都要先看它，而不是问别人：“页面怎么不动了?”

由于 JS 代码不是特别严谨，因此给出的错误提示也都很概括。我们可以做个对比：

- JSP 错误可以精确到某行。
- PHP 错误可以精确到某行。
- Rails 错误可以精确到某行。
- Vue.js、Angular、Titanium 等 JS 框架错误可以精确到某个文件。

这是由于所有的 JS 框架的表现层都是“框架怪胎”，是一种与 js 语言环境妥协的代码，出了问题很难定位到最底层的根源。而 JSP、PHP、Rails ERB 则是“正常框架”，出了问题可以直接找到最底层。

因此我们要有一定的经验 Debug，来理解错误日志。如图 5-11 所示的 Vue.js 框架报错信息。

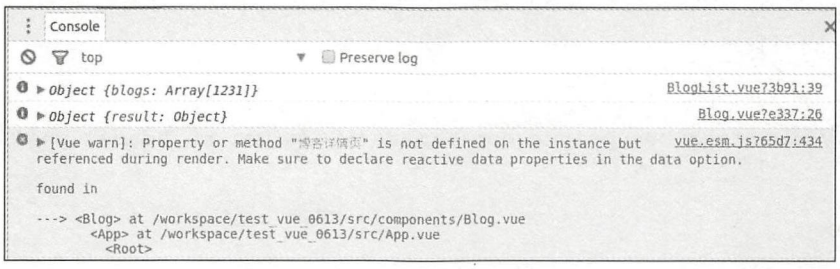


图 5-11　Vue.js 框架报错信息

```
vue.esm.js?65d7:434 [Vue warn]: Property or method "博客详情页" is not defined
on
the instance but referenced during render. Make sure to declare reactive data
properties in the data option.

found in

---> <Blog> at /workspace/test_vue_0613/src/components/Blog.vue
     <App> at /workspace/test_vue_0613/src/App.vue
       <Root>
```

- vue.esm.js?65d7:434 表示错误的来源。这个文件一般人不知道来自哪里，我们暂且认为它来自临时产生的文件或虚拟 js 机。
- Property or method "博客详情页" is not defined ... 这句话提示了错误的原因。
- found in ... <Blog> at ... 这里则是调用栈，可以看出文件是从下调用到最上面的，问题出在最上面的文件，但是并没有给出错误的行数。

5.3.3　查看页面给出的错误提示

页面给出的错误提示如图 5-12 所示。

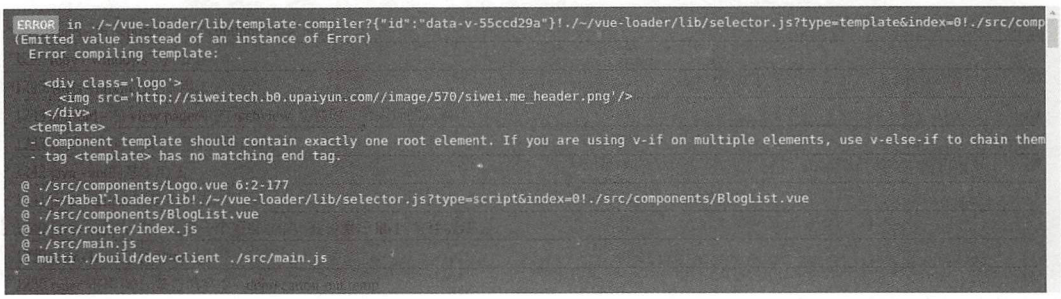


图 5-12　本地开发服务器的报错信息

```
 Error compiling template:


   <div class='logo'>
     <img
src='http://siweitech.b0.upaiyun.com//image/570/siwei.me_header.png'/>
   </div>
  <template>
  <script>
  </script>
  - Component template should contain exactly one root element. If you are
using v-if
  on multiple elements, use v-else-if to chain them instead.
  - Templates should only be responsible for mapping the state to the UI. Avoid
  placing tags with side-effects in your templates, such as <script>, as they
  will not be parsed.
  - tag <template> has no matching end tag.
```

这里 Error compiling template: 给出了提示，错误是在模板被编译时产生的。

下面给出的 HTML 代码是出错的点。

```
@ ./src/components/Logo.vue 6:2-177
@ ./~/babel-loader/lib!./~/vue-
loader/lib/selector.js?type=script&index=0!./src/components/BlogList.vue
@ ./src/components/BlogList.vue
@ ./src/router/index.js
@ ./src/main.js
@ multi ./build/dev-client ./src/main.js
```

这里是调用栈，可以看到@./src/components/Logo.vue 6:2-177，错误在 Logo.vue，第 6 行第 2 列。

5.4 基本命令

接下来要讲解的命令都是 vue-cli 提供的，可以认为是 Webpack+Vue.js 的结合。

5.4.1 建立新项目

```
$ vue init webpack my_blog
```

vue init 命令会创建一个文件夹。具体的说明可参见 3.6 Webpack 下的 Vue.js 项目文件结构。

5.4.2 安装所有的第三方包

```
$ npm install
```

该命令是根据 "package.json" 文件中定义的内容来安装所有用到的第三方 js 库。 所有的文件都会安装到 "node_module"目录下。

还可以输入--verbose 命令查看运行细节。

```
$ npm install --verbose
```

运行结果如下：

```
$ npm install --verbose
npm info it worked if it ends with ok
npm verb cli [ 'D:\\nodejs\\node.exe',
npm verb cli   'D:\\nodejs\\node_modules\\npm\\bin\\npm-cli.js',
npm verb cli   'install',
npm verb cli   '--verbose' ]
npm info using npm@6.1.0
npm info using node@v10.5.0
npm verb npm-session d1e752145cbb60ba
npm info lifecycle test_vue_0613@1.0.0~preinstall: test_vue_0613@1.0.0
npm timing stage:loadCurrentTree Completed in 1609ms
npm timing stage:loadIdealTree:cloneCurrentTree Completed in 12ms
npm timing stage:loadIdealTree:loadShrinkwrap Completed in 681ms
...
```

这样，出现问题时就很容易知道“卡”在哪里了。

5.4.3 在本地运行

使用下列命令：

```
$ npm run dev
```

默认会在 localhost 的 8080 端口启动一个小型的 Web 服务器，性能可以完全满足开发使用，还具备代理转发等功能。

源代码发生改变时，服务器也会自动重启（偶尔需要手动重启）。

代码如下所示：

```
dashi@i5-16g MINGW64 /d/workspace/vue_js_lesson_demo (master)
$ npm run dev

> test_vue_0613@1.0.0 dev D:\workspace\vue_js_lesson_demo
> node build/dev-server.js

[HPM] Proxy created: /api  ->  http://siwei.me
[HPM] Proxy rewrite rule created: "^/api" ~> ""
> Starting dev server...
 DONE  Compiled successfully in 2373ms10:55:18

> Listening at http://localhost:8080

 WAIT  Compiling...11:02:14

 DONE  Compiled successfully in 213ms11:02:14

 WAIT  Compiling...11:06:09

 DONE  Compiled successfully in 117ms11:06:09

 WAIT  Compiling...11:06:26

 DONE  Compiled successfully in 103ms11:06:26
 ...
```

5.4.4　打包编译

```
$ npm run build
```

该命令用于把 src 目录下的所有文件打包成 Webpack 的标准文件，供部署使用。 具体内容可参见 5.1 打包和部署。

第 6 章

◂ 进阶知识 ▸

虽然是进阶知识，但也是必学内容。

6.1 js 的作用域与 this

无论是 JavaScript 还是 emscript，变量的作用域都属于高级知识。我们想考查一个 js 程序员的水平如何，可以直接用作用域进行提问。

6.1.1 作用域

无论是 JavaScript 还是 emscript，对于作用域的使用基本相同，后者更加严密一些。

1. 全局变量可以直接引用

```
//全局变量 a:
var a = 1;
function one() {
  console.info(a)
}
打印结果是 1
```

2. 函数内的普通变量

```
function two(a ){
  console.info('a is' + a)
}
two(2)打印结果是 a is 2
```

3. 普通函数可以对全局变量做赋值

```
var a = 1;
function four(){
  console.info(' in four, before a=4: ' + a)
  if(true) a = 4;
  console.info(' in four, after a=4: ' + a)
}
```

运行结果如下：

```
four(4)
in four, before a=4: 1      ( 这个是符合正常的 scope 逻辑的。)
in four, after a=4: 4       ( 这个也是符合)
```

再次运行 console.info(a)，输出结果为 4，说明全局变量 a 在 four()函数中已经发生了永久的变化。

4. 通过元编程定义的函数

```
var six = ( function(){
  var foo = 6;
  return function(){
    return foo;
  }
}
)();
```

在上述代码中，js 解析器会先运行（忽略最后的()）。

```
var temp = function(){
  var foo = 6;
  return function(){
    return foo;
  }
}
```

然后运行 var six = (temp)()，six 就是：

```
function(){
    return foo;
}
```


上面的 foo 是来自方法最开始定义的 var foo=6，而这个变量的定义，是在一个 function() 中的。它不是一个全局变量。

如果在 console 中输入 foo，就会看到报错信息。

```
Uncaught ReferenceError: foo is not defined
```

5. 通过元编程定义的函数中的变量，不会污染全局变量

```
var foo = 1;

var six = ( function(){
  var foo = 6;
  return function(){
    console.info("in six, foo is: " + foo);
  }
}
)();
```

在上面的代码中，我们先定义了一个全局变量 foo，然后定义了一个方法 six，其中定义了一个临时方法 foo，并进行了一些操作。

运行：

```
six()   // 返回:   in six, foo is: 6
foo     // 返回: 1
```

6.1.2 this

对于 this 的使用，https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/this 指出了对于 JavaScript 中 this 的详细用法。在 emscript 中也一样。

简单地说，大家只要记住 this 是指当前作用域的对象实例就可以了。

```
var apple = {
  color: 'red',
  show_color: function() {
    return this.color
  }
}
```

输入 apple.show_color 即可看到输出 red，这里的 this 指的就是 apple 变量。

6.1.3　实战经验

1. 在 Vue 的方法定义中容易用错

当代码看起来没问题，但 console 总报“xx undefined”时，十有八九是忘记加 this 了。例如：

```
<html>
<head>
    <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
</head>
<body>
    <div id='app'>

        <br/>
        <button v-on:click='highlight' style='margin-top: 50px'>真的吗</button>
    </div>

    <script>
        var app = new Vue({
            el: '#app',
            data: {
                message: 'this 是很重要的。 不要忘记它哟~'
            },
            methods: {
                highlight: function() {
                    // 报错:  message is not defined.
                    message += '是的， 工资还会涨!'

                    // 正确的代码应该是:
                    // this.message += '是的， 工资还会涨!'
                }
            }

        })
    </script>
</body>
```

```
</html>
```

使用浏览器加载上述代码，我们会发现报错，如图 6-1 所示的浏览器报错 is not defined。

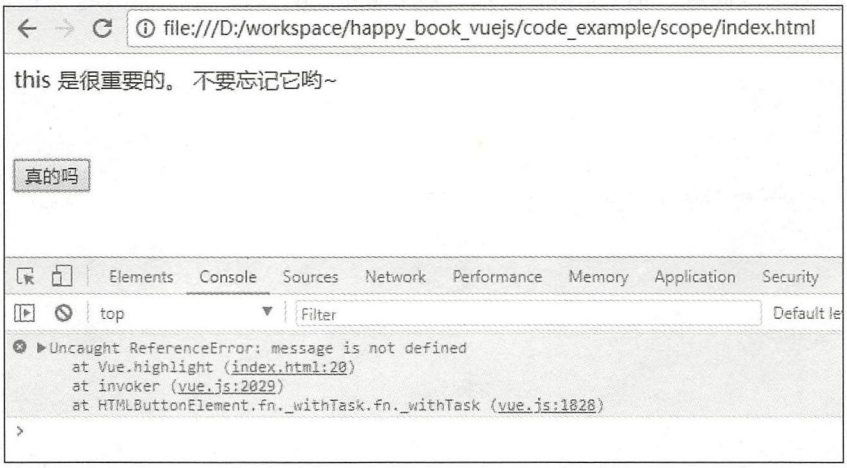


图 6-1　浏览器报销 is not defined

在 message += ...一行代码中，message 是当前 vue 实例的一个"property"（属性），如果希望在 methods 中引用这个属性，就需要使用 this.message 才行。这里的 this 对应的就是 var app = new Vue() 中定义的 app。

2. 在发起 HTTP 请求时容易用错

我们看下面的例子，是一段代码片段。

```
new Vue({
    data: {
        cities: [...]
    },
    methods: {
        my_http_request: function(){
            let that = this
            axios.get('http://mysite.com/my_api.do')
            .then(function(response){
                // 这里不能使用 this.cities 赋值
                that.cities = response.data.result
            })
        }
    }
```

```
})
```

在上面的代码中，定义了一个 cities 属性和一个 my_http_request 方法。my_http_request 方法会向远程发起一个请求，然后把返回的 response 中的值赋给 cities。

通过上面的代码可知，需要先在 axios.get 之前定义一个变量 let that=this。此时，this 和 that 都处于"Vue"的实例中。

但是，在 axios.get(..).then()函数中，就不能再使用 this 了。因为在 then(...)中，这是 function callback，其中的 this 会代表这个 HTTP request event。

注意，如果使用了 emscript 的=>，就可以避免上述问题。

3. 在 event handler 中容易用错

道理同上。

6.2 Mixin

Mixin 是一种更好的代码复用模式。

我们知道 Java、Object C 中的 interface、implements、extends 等关键字的意义是为了让代码可以复用、继承。但是这几种方法理解起来很不直观，给人一种模糊的感觉，特别是不习惯“设计模式”的用户。

在 js、Ruby 等动态语言中，如果需要复用代码，可直接使用 mixin。

1. Mixin 的概念

Mixin 实际上是利用语言的特性（关键字），以更加简洁易懂的方式实现了“设计模式”中的“组合模式”。可以定义一个公共的类，这个类叫做 mixin，然后让其他的类通过 include 的语言特性来包含 mixin，直接具备了 mixin 的各种方法。

下面看一下在 Vue.js 中如何使用 mixin 这种强大的工具。

2. 建立一个 Mixin 文件

可以在 src/mixin 目录下创建，例如 src/mixin/common_hi.js 文件：

```
export default {
  methods: {
    hi: function(name){
      return "你好, " + name;
    }
  }
}
```

3. 使用

Mixin 使用起来十分简单，在对应的 js 文件，或者 vue 文件的<script>代码中引用即可。例如，新建一个 vue 文件：src/components/SayHiFromMixin.vue，内容如下：

```
<template>
  <div>
    {{hi("from view")}}
  </div>
</template>

<script>
import CommonHi from '@/mixins/common_hi.js'
export default {
  mixins: [CommonHi],
  mounted() {
    alert( this.hi('from script code'))
  }
}
</script>
```

注意：

- mixins: [CommonHi]中的中括号表示数组。
- 在 mounted()中调用的话，需要带有 this 关键字，如 this.hi()。

路由如下：

```
import SayHiFromMixin from '@/components/SayHiFromMixin'

export default new Router({
  routes: [
    {
      path: '/say_hi_from_mixin',
      name: 'SayHiFromMixin',
      component: SayHiFromMixin
    }
  ]
} )
```

运行结果如图 6-2 所示的调用例子。

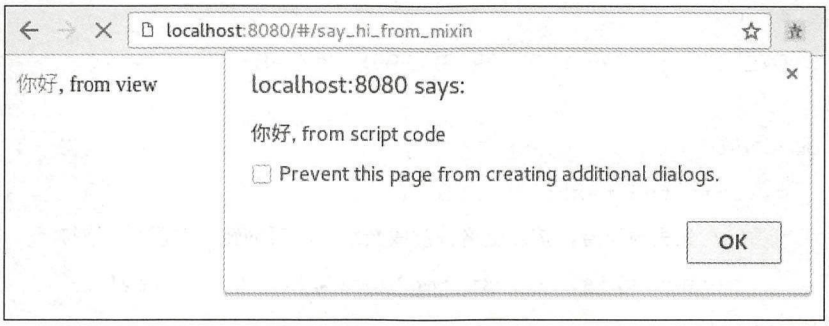


图 6-2　Mix 的调用例子

6.3 使用 Computed Properties（计算得到的属性）和 watchers（监听器）

我们想要在页面上显示某个变量的值时，必须经过一些计算。例如：

```
<div id="example">
  {{ some_string.split(',').reverse().join('-') }}
</div>
```

越是复杂，后期就越容易出错。此时，我们需要一种机制，可以方便地创建通过计算得来的数据，Computed Properties 就是我们的解决方案。

6.3.1 典型例子

```
<html>
<head>
    <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
</head>
<body>
    <div id='app'>
        <p> 原始字符串：  </p>
        <p> 通过运算后得到的字符串：  </p>
    </div>
    <script>
        var app = new Vue({
```

```
        el: '#app',
        data: {
            my_text: 'good good study, day day up'
        },
        computed: {
            my_computed_text: function(){
                // 先去掉逗号，按照空格分割成数组，然后翻转，并用'-'连接
                return this.my_text.replace(',', '').split('
').reverse().join('-')
            }
        }

    })
  </script>
</body>
</html>
```

上面的关键代码是在 Vue 的构造函数中传入一个 Computed 的段落。

使用浏览器运行后，可以看到如图 6-3 所示的使用 Computed Properties 翻转字符串。

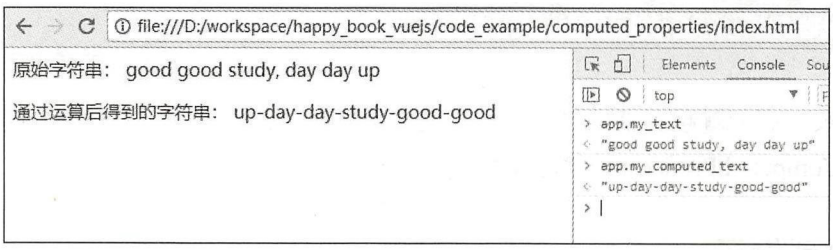


图 6-3　使用 Computed Properties 翻转字符串

也可以打开 console 进行查看。输入：

```
> app.my_text
```

得到"good good study, day day up"。输入：

```
> app.my_computed_text
```

得到转换后的"up-day-day-study-good-good"。

6.3.2　Computed Properties 与 普通方法的区别

根据上面的例子，我们可以使用普通方法来实现。

```
<html>
<head>
    <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
</head>
<body>
    <div id='app'>
        <p> 原始字符串：  </p>
        <p> 通过运算后得到的字符串：  {{my_computed_text() }} </p>
    </div>
    <script>
        var app = new Vue({
            el: '#app',
            data: {
                my_text: 'good good study, day day up'
            },
            methods: {
                my_computed_text: function(){
                    return this.my_text.replace(',', '').split('
').reverse().join('-') + '，我来自于 function，不是 computed '
                }
            }
        })
    </script>
</body>
</html>
```

运行上面的代码后，如图 6-4 所示的通过某个普通函数翻转字符串。

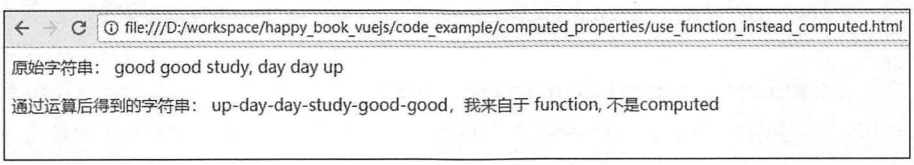


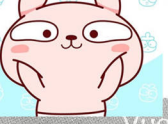
图 6-4　通过某个普通函数翻转字符串

可以发现两者达到的效果是一样的。

区别在于：使用 Computed Properties 的方式，会把结果“缓存”起来，每次调用对应的 Computed Properties 时，只要对应的依赖数据没有改动，那么就不会变化；使用 function 实现的版本，则不存在缓存问题，每次都会重新计算对应的数值。因此，我们需要按照实际情况，选择使用 Computed Properties 还是普通 function 的形式。

6.3.3　watched property

Vue.js 中的 property（属性）是可以根据计算发生变化（computed），或者根据监听



（watch）其他变量的变化而发生变化的。

下面看一下，如何根据监听（watch）其他的变量而自身发生变化的例子。

```
<html>
<head>
   <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
</head>
<body>
   <div id='app'>
      <p>
         我所在的城市： <input v-model='city' /> (这是个 watched property)
      </p>
      <p>
         我所在的街道： <input v-model='district' /> (这是个 watched property)
      </p>
      <p> 我所在的详细一些的地址：  (每次其他两个发生变化，这里就会跟着变化) </p>
   </div>
   <script>
      var app = new Vue({
         el: '#app',
         data: {
            city: '北京市',
            district: '朝阳区',
            full_address: "某市某区"
         },
         watch: {
            city: function(city_name){
               this.full_address = city_name + '-' + this.district
            },
            district: function(district_name){
               this.full_address = this.city + '-' + district_name
            }
         }

      })
   </script>
</body>
</html>
```

在上面的代码中，watch: { city: ..., district: ...}表示 city 和 district 已经被监听了， 这两个都是 watched properties。只要 city 和 district 发生变化，full_address 就会随之变化。

使用浏览器打开上面的代码，此时由于 city 和 district 没有发生变化，因此 full_address 的值还是 "某市某区"。如图 6-5 所示为城市和街道变化前的页面。

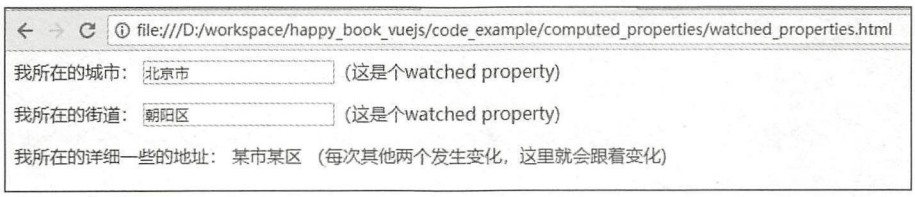


图 6-3　城市和街道变化前的页面

当在“我所在街道：”的文本框中添加“望京街道”后，可以看到下面的“我所在的详细一些的地址：”发生了变化。如图 6-6 所示为城市和街道变化后的页面。

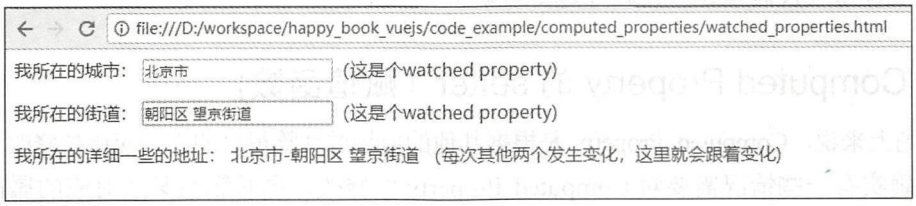


图 6-6　城市和街道变化后的页面

使用 computed 会比 watch 更加简洁。

上面的例子，我们也可以使用 computed 来改写。

```
<html>
<head>
    <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
</head>
<body>
    <div id='app'>
        <p>
            我所在的城市： <input v-model='city' />
        </p>
        <p>
            我所在的街道： <input v-model='district' />
        </p>
        <p> 我所在的详细一些的地址：  （这是使用 computed 实现的版本） </p>

    </div>
    <script>
        var app = new Vue({
            el: '#app',
            data: {
                city: '北京市',
                district: '朝阳区',
            },
            computed: {
                full_address: function(){
```

```
                return this.city + this.district;
            }
        }
    })
</script>
</body>
</html>
```

可以看到方法少了一个，data 中定义的属性也少了一个，简洁了不少。代码简洁，维护起来就会很容易（代码量越少，程序越好理解）。

6.3.4 Computed Property 的 setter（赋值函数）

从原则上来说，Computed Property 是根据其他的值经过计算得来的，不应该被修改。不过在开发中，确实有一些情况需要对 Computed Property 做修改，同时影响某些对应的属性（过程与上面是相反的）。

我们看下面的代码。

```
<html>
<head>
    <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
</head>
<body>
    <div id='app'>
        <p>
            我所在的城市：  <input v-model='city' />
        </p>
        <p>
            我所在的街道：  <input v-model='district' />
        </p>
        <p> 我所在的详细一些的地址：     <input v-model='full_address' /> </p>

    </div>
    <script>
        var app = new Vue({
            el: '#app',
            data: {
                city: '北京市',
                district: '朝阳区',
            },
            computed: {
                full_address: {
                    get: function(){
                        return this.city + "-" + this.district;
```

```
                },
                set: function(new_value){
                    this.city = new_value.split('-')[0]
                    this.district = new_value.split('-')[1]
                }
            }
        }
    })
</script>
</body>
</html>
```

在上面的代码中有这样一段：

```
computed: {
    full_address: {
        get: function(){
            return this.city + "-" + this.district;
        },
        set: function(new_value){
            this.city = new_value.split('-')[0]
            this.district = new_value.split('-')[1]
        }
    }
}
```

可以看出，上面的 get 代码段就是原来的代码内容。 而 set 端中，则定义了如果 Computed Property（也就是 full_address）发生变化时，city 和 district 的值应该如何变化。

使用浏览器打开后，在“我所在的详细一些的地址：”中输入一些文字，可以看到对应的“我所在的街道”发生了变化。如图 6-7 所示使用 setter 修改 computed properties。

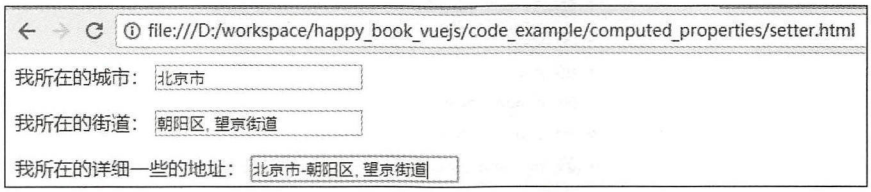


图 6-7　使用 setter 修改 computed properties

6.4 Component（组件）进阶

在 Web 开发中，Component 是十分常见的，只要是生产环境的项目，就一定会有 Component。

6.4.1 实际项目中的 Component

下面是一个实际项目中的例子，该项目只做了两个月，其中就发展到了 32 个 Component。如图 6-8 所示实际项目中复杂的 component 文件结构。

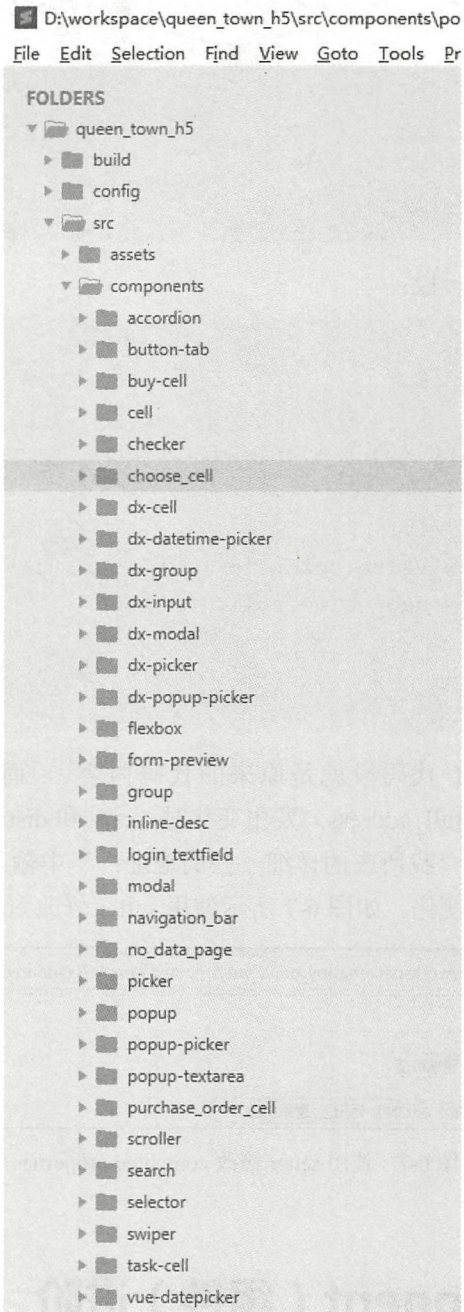


图 6-6 实际项目中复杂的 component 文件结构

很多时候，一个 Component 中嵌套着另一个，这个 component 再嵌套另外 5 个。例如：popup-picker 这个 Component 中，看起来是这样的：

```
<template>
  <div>
     <popup
        class="vux-popup-picker"
        :id="`vux-popup-picker-${uuid}`"
        @on-hide="onPopupHide"
        @on-show="onPopupShow">

        <picker
           v-model="tempValue"
           @on-change="onPickerChange"
           :columns="columns"
           :fixed-columns="fixedColumns"
           :container="'#vux-popup-picker-'+uuid"
           :column-width="columnWidth">
        </picker>
     </popup>
  </div>
</template>
<script>
import Picker from '../picker'
import Popup from '../popup'
...
</script>
```

可以看到，这个 Component 中还包含另外两个，即 popup 和 picker。

这个时候，新人往往会眼花缭乱，如果看到 this.$emit，就更晕了。因此，要做好实际项目，一定要学好本章内容。

Component 命名规则

官方建议每个 Component 的命名都使用小写字母+横线的形式。例如：

```
Vue.component('my-component-name', { /* ... */ })
```

这个是符合 W3C 规范的。也可以定义为：

```
Vue.component('MyComponentName', { /* ... */ })
```

可以使用<MyComponentName/>调用，也可以使用 <my-component-name/>调用。

6.4.2 Prop

1. Prop 命名规则

Prop 命名规则同 component，建议使用小写字母 +'-' 连接。

Prop 有多种类型，包括字符串、数字、bool、数组和 Object。例如：

```
props: {
  title: "Triple Body",
  likes: 38444,
  isPublished: true,
  commentIds: [30, 54, 118, 76],
  author: {
     name: "Liu Cixin",
     sex: "male"
  }
}
```

2. 可以动态为 prop 赋值

下面是一个静态的赋值。

```
<blog-post title="Vue.js 的学习笔记"></blog-post>
```

下面是一个动态的赋值。

```
// 1. 在 script 中定义
post = {
   title: 'Triple body',
   author: {
      name: "Big Liu",
      sex: 'male'
   }
}

// 2. 在模板中使用。
<blog-post v-bind:title="post.title + 'by' + post.author.name"></blog-post>
```

赋值时，只要符合标准的类型，都可以传入（包括 String、bool、array 等）。

3. 使用 Object 为 Prop 赋值

假设定义有：

```
post = {
   author: {
```

```
        name: "Big Liu",
        sex: 'male'
    }
}
```

那么下面的代码：

```
<blog-post v-bind:author></blog-post>
```

等价于：

```
<blog-post v-bind:name="author.name" v-bind:sex="author.sex"></blog-post>
```

4. 单向的数据流

当“父页面”引用一个“子组件”时，如果“父页面”中的变量发生了变化，那么对应的“子组件”也会发生页面的更新，反之则不行。

5. Prop 的验证

Vue.js 的组件 Prop 是可以被验证的。 如果验证不匹配，浏览器的 console 就会弹出警告（warning），这个对于开发非常有利。

例如下面的代码：

```
Vue.component('my-component', {
    props: {
        name: String,
        sexandheight: [String, Number],
        weight: Number,
        sex: {
            type: String,
            default: 'male'
        }
    }
})
```

其中，name 必须是字符串，sexandheight 必须是数组。第一个元素是 String，第二个元素是 Number，weigh 必须是 Number， sex 是 String，默认值是 'male'。

Prop 支持的类型有 String、Number、Boolean、Array、Object、Date、Function、Symbol 等。

6. Non Prop（非 Prop）的属性

很多时候，因为 Component 的作者无法预见应该用哪些属性， 所以 Vue.js 在设计时，支持让 Component 接受一些没有预先定义的 prop。例如：

```
Vue.component('my-component', {
```

```
    props: ['title']
})

<my-component title='三体' second-title='第二册： 黑暗森林'></my-component>
```

上面代码中的 title 就是预先定义的 "Prop"，second-title 就是“非 Prop”。

如果想传递一个 non-pro，非常简单，prop 怎么传 non-prop 就怎么传。

6.4.3 Attribute

1. Attribute 的合并和替换

如果 component 中定义了一个 attribute，例如：

```
<template>
    <div color="red">我的最终颜色是蓝色</div>
</template>
```

如果在引用了这个“子组件”的“父页面”中也定义了同样的 attribute，例如：

```
<div>
    <my-component color="blue"></my-component>
</div>
```

那么“父页面”传递进来的 color="blue"就会替换“子组件”中的 color="red"。

但是，对于 class 和 style 是例外的。上面的例子中，如果将 attribute 换成 class，那么最终 component 中 class 的值就是 "red blue"（发生了合并）。

5. 避免子组件的 attribute 被父页面影响

根据以上的分析，我们知道“父页面”的值总会替换“子组件”中的同名 attribute。如果不希望有这样的情况发生，就可以在定义 component 时，这样做：

```
Vue.component('my-component', {
  inheritAttrs: false,
  // ...
})
```

6.5 Slot

作为对 Component 的补充，Vue.js 增加了 Slot 功能。

6.5.1　普通的 Slot

我们通过具体的例子来说明。

```
<html>
<head>
   <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
</head>
<body>
   <div id='app'>
      <study-process>
         我学习到了 Slot 这个章节
      </study-process>
   </div>
   <script>
      Vue.component('study-process', {
        data: function () {
          return {
            count: 0
          }
        },
        template: '<div><slot></slot></div>'
      })
      var app = new Vue({
         el: '#app',
         data: {
         }
      })
   </script>
</body>
</html>
```

从上面的代码中可以看到，我们先定义了一个 Component。

```
Vue.component('study-process', {
  data: function () {
    return {
      count: 0
    }
  },
  template: '<div><slot></slot></div>'
})
```

在 Component 的 template 中，是这样的：

```
template: '<div><slot></slot></div>'
```

这里就是我们定义的 slot。在调用 Component 时：

```
<study-process>
   我学习到了 Slot 这个章节
</study-process>
```

“我学习到了 Slot 这个章节”就好像一个参数一样传入到了 Component 中。Component 发现自身已经定义了 slot，就会把这个字符串放到 slot 的位置并显示出来。

如图 6-9 所示的使用 Slot 的页面。

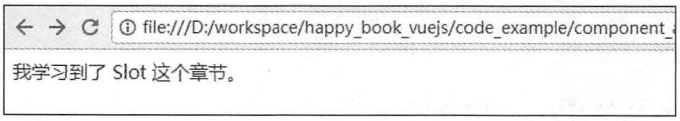


图 6-9　使用 Slot 的页面

6.5.2　named slot

named slot 也就是带有名字的 slot，很多时候我们可能需要多个 slot。来下面的例子：

```
<html>
<head>
   <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
</head>
<body>
   <div id='app'>
      <study-process>
         <p slot='slot_top'>
            Vue.js 比起别的框架真的简洁好多
         </p>

            我学习到了  Slot 这个章节

         <h5 slot='slot_bottom'>
            再也不怕 H5 项目了
         </h5>
      </study-process>
```

```
    </div>
    <script>
        Vue.component('study-process', {
          template: '<div>' +
          '<slot name="slot_top"></slot>' +
          '<slot></slot>' +
          '<slot name="slot_bottom"></slot>' +
          '</div>'
        })
        var app = new Vue({
            el: '#app',
            data: {
            }
        })
    </script>
</body>
</html>
```

在上面的代码中，我们定义了这样的 component：

```
Vue.component('study-process', {
  template: '<div>' +
  '<slot name="slot_top"></slot>' +
  '<slot></slot>' +
  '<slot name="slot_bottom"></slot>' +
  '</div>'
})
```

其中，<slot name="slot_top"></slot> 就是一个 named slot（具备名字的 slot）。这样，在后面对于 component 的调用中：

```
<p slot='slot_top'>
    Vue.js 比起别的框架真的简洁好多
</p>
```

就会渲染在对应的位置了。

6.5.3　slot 的默认值

我们可以为 slot 加上默认值，这样当“父页面”没有指定某个 slot 时，就会显示这个默

认值了。例如：

```
<slot name="slot_top">这里 top slot 的默认值 </slot>
```

6.6 Vuex

Vuex 是状态管理工具，与 React 中的 Redux 相似，但是更加简洁、直观。

简单地说，Vuex 可以帮我们管理“全局变量”，供任何页面在任何时候使用。与其他语言中的“全局变量”相比，Vuex 的优点如下。

（1）Vuex 中的变量状态是响应式的。当某个组件读取该变量时，只要 Vuex 中的变量发生变化，对应的组件就会发生变化（类似于双向绑定）。

（2）用户或程序无法直接改变 Vuex 中的变量，必须通过 Vuex 提供的接口来操作，该接口就是通过"commit mutation"实现的。

Vuex 非常重要，不管是大项目还是小项目都会用到它，我们必须会用。完整的官方文档可参见：

```
https://vuex.Vue.js.org/zh-cn/getting-started.html
```

Vuex 的内容很庞大，用到了比较“烧脑”的设计模式（这是由于 JavaScript 语言本身不够严谨和成熟决定的），因此笔者不打算把源代码和实现原理详细讲一遍，大家只要熟练使用就可以了。

6.6.1 正常使用的顺序

假设有两个页面：页面 1 和页面 2 共同使用一个变量 counter。页面 1 对 "counter" + 1 后，页面 2 的值也会发生变化。

1. 修改 package.json

增加 vuex 的依赖声明，代码如下：

```
"dependencies": {
  "vuex": "^2.3.1"
},
```

如果不确定 vuex 用哪个版本，就先手动安装一下。

```
$ npm install vuex --verbose
```

然后看安装的版本号就可以了。

2. 新建 store 文件

文件名：src/vuex/store.js。这个文件的作用是在整个 Vue.js 项目中声明：我们要使用 Vuex 进行状态管理。

文件内容如下：

```
import Vue from 'vue'
import Vuex from 'vuex'

// 这个就是我们后续会用到的 counter 状态
import counter from '@/vuex/modules/counter'

Vue.use(Vuex)

const debug = process.env.NODE_ENV !== 'production'
export default new Vuex.Store({
    modules: {
          counter                    // 所有要管理的 module 都列在这里
    },
    strict: debug,
    middlewares: []
})
```

在上面代码中，大部分是“鸡肋”代码。有用的代码如下：

```
import counter from '@/vuex/modules/counter'
...
    modules: {
        counter
    }
...
```

这里定义了所有的 vuex module。

3. 新建 vuex/module 文件

文件名： src/vuex/modules/counter.js。内容如下：

```
import { INCREASE } from '@/vuex/mutation_types'

const state = {
  points: 10
}
```

```
const getters = {
  get_points: state => {
    return state.points
  }
}

const mutations = {
  [INCREASE](state, data){
    state.points = data
  }

}

export default {
  state,
  mutations,
  getters
}
```

上面是一个典型的 vuex module，其作用就是计数。

- state：表示状态，可以认为 state 是一个数据库，保存了各种数据，但无法直接访问里面的数据。
- mutations：表示变化，可以认为所有的 state 都是由 mutation 来驱动变化的，也可以认为它是 setter。
- getter：取值的方法，与 setter 相对。

如果希望获取某个数据，就需要调用 vuex module 的 getter 方法；如果希望更改某个数据，就需要调用 vuex module 的 mutation 方法。

4. 新增文件：src/vuex/mutation_types.js

```
export const INCREASE = 'INCREASE'
```

大家在做项目时，要统一把 mutation type 定义在这里，类似于方法列表。

这个步骤不能省略，Vue.js 官方也建议这样写。好处是维护时可以看到某个 mutation 有多少种状态。

5. 新增路由：src/routers/index.js

```
import ShowCounter1 from '@/components/ShowCounter1'
import ShowCounter2 from '@/components/ShowCounter2'
```

```
export default new Router({
  routes: [
    {
      path: '/show_counter_1',
      name: 'ShowCounter1',
      component: ShowCounter1
    },
    {
      path: '/show_counter_2',
      name: 'ShowCounter2',
      component: ShowCounter2
    }
    ]
})
```

6. 新增两个页面：src/components/ShowCounter1.vue 和 src/components/ShowCounter2.vue

这两个页面基本相同。

```
<template>
  <div>
    <h1> 这个页面是 1 号页面 </h1>
     <br/>
    <input type='button' @click='increase' value='点击增加 1'/><br/>
    <router-link :to="{name: 'ShowCounter2'}">
         计数页面 2
    </router-link>
  </div>
</template>

<script>
import store from '@/vuex/store'
import { INCREASE } from '@/vuex/mutation_types'
export default {
  computed: {
    points() {
      return store.getters.get_points
```

```
    }
  },
  store,
  methods: {
    increase() {
      store.commit(INCREASE, store.getters.get_points + 1)
    }
  }
}
</script>
```

我们可以在<script>中调用 vuex 的 module 方法。例如：

```
increase() {
    store.commit(INCREASE, store.getters.get_points + 1)
}
```

store.getters.get_points 就是通过 getter 获取到状态 "points" 的方法。store.commit(INCREASE, ..) 则是通过 INCREASE 这个 action 来改变 "points"的值。

6.6.2 Computed 属性

Computed 代表的是某个组件（component）的属性，该属性是计算出来的。每当计算因子发生变化时， 这个结果也要重新计算。

下面的代码中：

```
<script>
export default {
  computed: {
    points() {
      return store.getters.get_points
    }
  },
</script>
```

就是定义了一个叫作 "points" 的 "computed" 属性。然后在页面中显示这个“计算属性”：

```
<template>
  <div>
```

```
    </div>
</template>
```

就可以把 state 中的数据显示出来，并自动更新。

重启服务器（$ npm run dev）之后运行，可以看到如图 6-10 所示的使用 Vuex 实现的计数器。单击按钮，计数器的数值就会加 1。

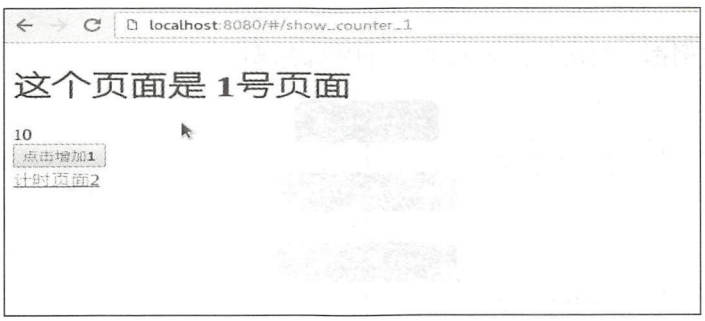


图 6-10　使用 Vuex 实现的计数器

6.6.3　Vuex 原理图

如图 6-11 所示的 Vuex 原理图。

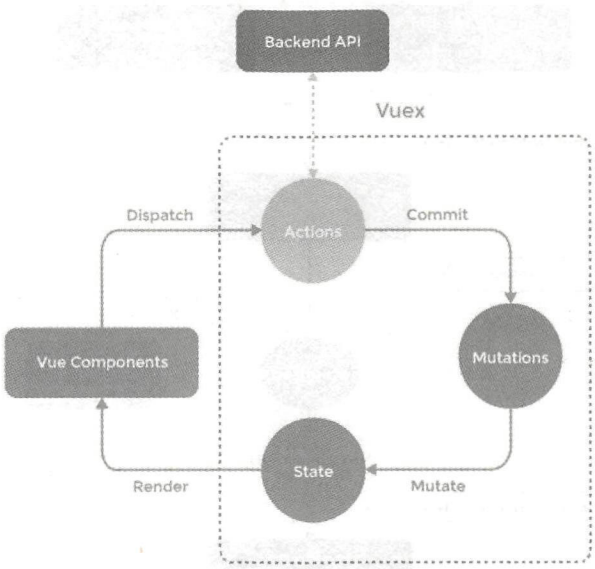


图 6-9　Vuex 原理图

可以看到：

（1）总体包括 Action、Mutation 和 State 三个概念，State 由 Mutation 来变化。

（2）Vuex 通过 Action 与后端 API 进行交互。

（3）Vuex 通过 State 渲染前端页面。

（4）前端页面通过触发 Vuex 的 Action 来提交 mutation，以达到改变 "state"的目的。

6.7 Vue.js 的生命周期

每个 Vue.js 实例都会经历如图 6-12 所示的生命周期。

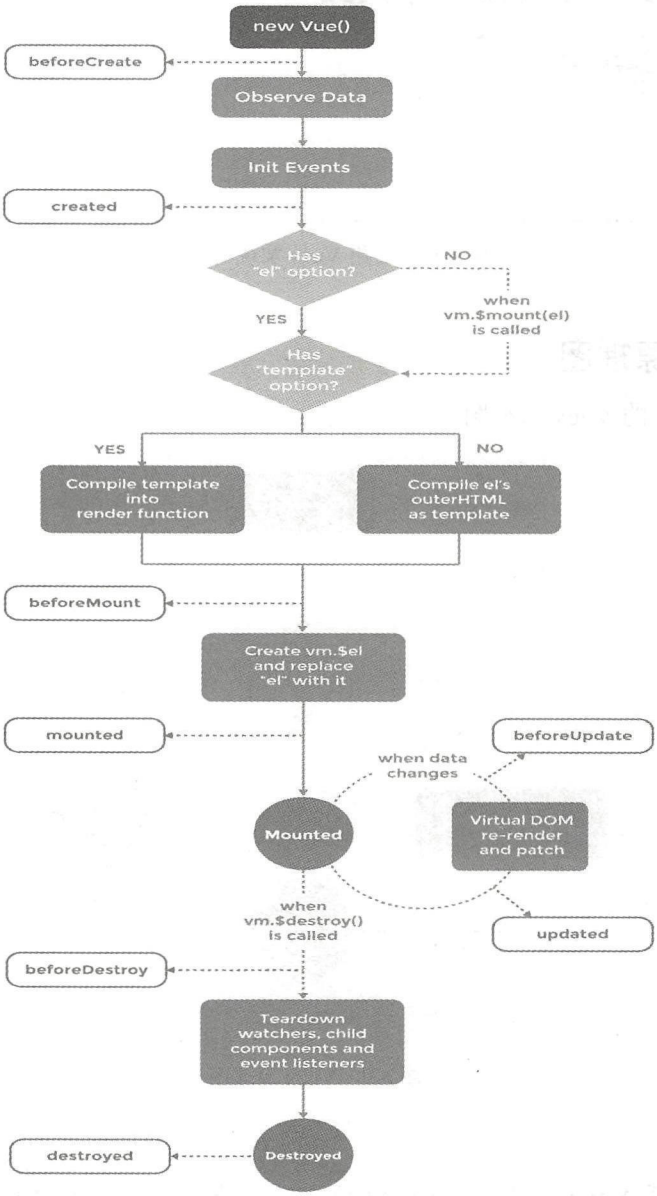


图 6-12　Vue.js 的生命周期

可以看出基本周期如下。

（1）created（创建好 DOM）。
（2）mounted（页面基本准备好了）。
（3）updated（update 可以理解为手动操作触发）。
（4）destroyed（销毁）。

上面周期中的（1）、（3）、（4）都是自动触发的，每一步都有对应的 beforeXyz 方法。因此，我们一般使用 mounted 作为页面初始化时执行的方法。

6.8 最佳实践

1. 适当地使用 vuex

能不用就不用，不要为了使用而使用，如一个小方法就可以“搞定”的事情，非要使用 5 个设计模式来实现。

6. 不要过度使用 CSS 框架

CSS 框架通常会增加文件体积，如 bootstrap、ele.me 前端框架，特别是使用 Android 中的 Webview 加载 H5 页面时，基本上 1k 的 CSS 就会消耗 1ms。

7. 使用 CDN 存放图片文件

upyun 就是一个不错的选择，阿里的 oss 也很好。

8. js、css 尽量使用压缩

让 js、css 都以 zip 的形式发送和接收，一般会减少 30% ~ 60%的体积和传送时间，具体可参考 nginx 文档。

9. 灵活使用第三方 Vue 插件

第三方 Vue 插件有轮播图、表单验证等。
好的程序员不一定算法好，但一定要对各种第三方插件特别熟知。

10. 前端逻辑务必简单

能在后台处理的，绝对不要放在前端处理，因为 Vue.js 擅长的不是处理数据结构。例如，前端需要展示一个列表，后端的接口就应该给出 JSON 中的数组，而不是给出一个字符串由前端去解析。

11. 不用写行末分号

Vue.js 源代码中没有一行有“行末分号”。

12. 灵活使用 CSS、HTML 预处理工具

我们知道 JADE、HAML 可以生成 HTML；SASS、SCSS、LESS 可以生成 CSS。如果公司的员工比较多，那么建议直接使用原生的 HTML、CSS；如果是一个人独立负责整个项目，那么用 JADE、SCSS 也没问题。

6.9 Event Handler 事件处理

Event Handler 之所以会被 Vue.js 放到很重要的位置，是基于以下考虑。

（1）把与事件相关的代码独立写出来，非常容易定位各种逻辑，维护起来也方便。

（2）event handler 被独立出来之后，页面的 DOM 元素看起来就会很简单，容易理解。

（3）当一个页面被关闭时，对应的 ViewModel 会被回收，该页面定义的各种 event handler 也会被一并垃圾回收，不会造成内存溢出。

6.9.1 支持的 Event

我们在前面曾经看到过 v-on:click，那么都有哪些事件可以被 v-on 支持呢？只要是标准的 HTML 定义的 Event，都可以被 Vue.js 支持，如 focus（元素获得焦点）、blur（元素失去焦点）、click（单击鼠标左键）、dblclick（双击鼠标左键）、contextmenu（单击鼠标右键）、mouseover（指针移到有事件监听的元素或其子元素内）、mouseout（指针移出元素或其子元素上）、keydown（键盘动作：按下任意键）及 keyup（键盘动作：释放任意键）。

可以在下面链接中查看所有 HTML 标准事件。

```
https://developer.mozilla.org/zh-CN/docs/Web/Events
```

一共定义了 162 个标准事件和几十个非标准事件，以及 Mozilla 的特定事件。如图 6-13 所示的 HTML 标准事件。

标准事件

这些事件在官方Web规范中定义，并且应在各个浏览器中通用。 每个事件都和代表事件接收方的对象（由此您可以查到每个事件提供的数据），定义这个事件的标准或标准链接会一起列出。

事件名称	事件类型	规范	触发时机...
abort	UIEvent	DOM L3	资源载入已被中止
abort	ProgressEvent	Progress and XMLHttpRequest	Progress 被终止(不是error造成的)
abort	Event	IndexedDB	事务已被中止
afterprint	Event	HTML5	相关文档已开始打印或打印预览已被关闭
animationend	AnimationEvent	CSS Animations	完成一个CSS 动画
animationiteration	AnimationEvent	CSS Animations	重复播放一个CSS 动画
animationstart	AnimationEvent	CSS Animations	一个CSS 动画已开始
audioprocess	AudioProcessingEvent	Web Audio API audioprocess	一个ScriptProcessorNode 的输入缓冲区可处理
audioend	Event	Web Speech API	用户代理捕捉到用以语音识别的音频
audiostart	Event	Web Speech API	用户代理开始捕捉用以语音识别的音频
beforeprint	Event	HTML5	相关文档将要开始打印或准备打印预览
beforeunload	BeforeUnloadEvent	HTML5	即将卸载 window，document 及其资源
beginEvent	TimeEvent	SVG	A SMIL animation element begins.
blocked_indexedDB	IDBVersionChangeEvent	IndexedDB	An open connection to a database is blocking a versionchange transaction on the same database.

图 6-13　HTML 标准事件

不用全部记住，在日常开发中只有十几个是常见的 event。

6.9.2　使用 v-on 进行事件绑定

我们可以认为，几乎所有的事件都是由 v-on 这个 directive 来驱动的。

1. 在 v-on 中使用变量

可以在 v-on 中引用变量，代码如下：

```
<html>
<head>
    <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
</head>
<body>
    <div id='app'>
```

```
        您单击了：{{ count }} 次
        <br/>
        <button v-on:click='count += 1' style='margin-top: 50px'> + 1</button>
    </div>

    <script>
        var app = new Vue({
            el: '#app',
            data: {
                count: 0
            }
        })
    </script>
</body>
</html>
```

使用浏览器打开后，单击 +1 按钮，就可以看到 count 变量会随之+1。如图 6-14 所示计数器每次单击都会加 1。

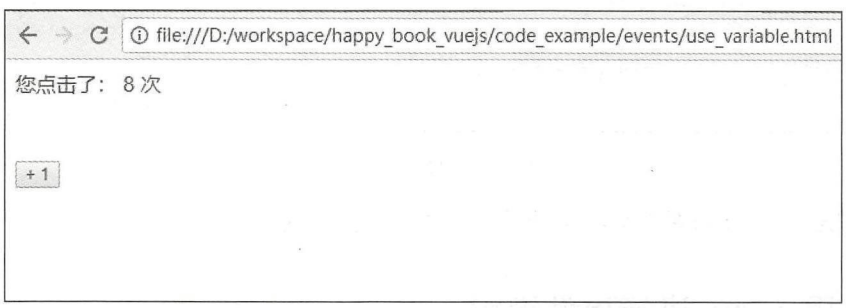


图 6-14　计数器每次单击都会加 1

2. 在 v-on 中使用方法名

上面的例子也可以通过下面的代码来实现。

```
<html>
<head>
    <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
</head>
<body>
    <div id='app'>
        您单击了：{{ count }} 次
```

```
        <br/>
        <button v-on:click='increase_count' style='margin-top: 50px'>  + 1
</button>
    </div>

    <script>
        var app = new Vue({
            el: '#app',
            data: {
                count: 0
            },
            methods: {
                increase_count: function(){
                    this.count += 1
                }
            }
        })
    </script>
</body>
</html>
```

可以看到，在 v-on:click='increase_count' 中， increase_count 就是一个方法名。

3. 在 v-on 中使用方法名 + 参数

也可以直接使用 v-on:click='some_function("your_parameter")' 这样的写法。例如：

```
<html>
<head>
    <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
</head>
<body>
    <div id='app'>
        {{ message }}
        <br/>
        <button v-on:click='say_hi("明日的 Vue.js 大神")' style='margin-top:
50px'> 跟我打个招呼~ </button>
    </div>
```

```
    <script>
        var app = new Vue({
            el: '#app',
            data: {
                message: "这是一个在 click 中调用方法 + 参数的例子"
            },
            methods: {
                say_hi: function(name){
                    this.message = "你好啊，" + name + "!"
                }
            }
        })
    </script>
</body>
</html>
```

使用浏览器打开后，单击按钮就可以看到结果。如图 6-15 所示的通过 click 事件来调用方法。

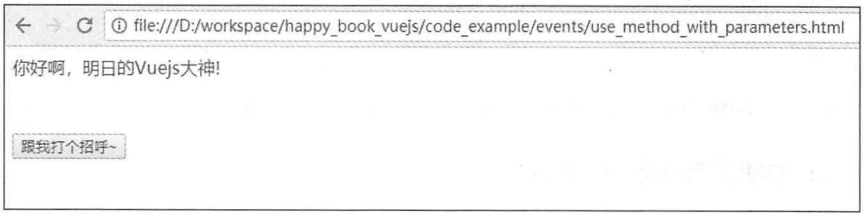


图 6-15　通过 click 事件来调用方法

4. 重新设计按钮的逻辑

在实际开发中往往会遇到这样的情况：单击某个按钮，或者触发某个事件后，希望停止按钮的默认动作。

例如，提交表单时，我们希望先对该表单进行验证，如果验证不通过，该表单就不提交。此时，如果希望表单不提交，就需要让 submit 按钮不进行下一步动作。在所有的开发语言中，都会有一个对应的方法叫作 "preventDefault"（停止默认动作）。

下面来看一个例子。

```
<html>
<head>
    <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
```

```
</head>
<body>
    <div id='app'>

        请输入您想打开的网址,        <br/>
        判断规则是:                  <br/>
        1. 务必以 "http://"开头      <br/>
        2. 不能是空字符串            <br/>
        <input v-model="url" placeholder="请输入 http:// 开头的字符串, 否则不会跳转"
/> <br/>
        <br/>
        <a v-bind:href="this.url" v-on:click='validate($event)'> 点我确定 </a>
    </div>

    <script>
        var app = new Vue({
            el: '#app',
            data: {
                url: ''
            },
            methods: {
                validate: function(event){
                    if(this.url.length == 0 || this.url.indexOf('http://') != 0){
                        alert("您输入的网址不符合规则。 无法跳转")
                        if(event){
                            alert("event is: " + event)
                            event.preventDefault()
                        }
                    }
                }
            }
        })
    </script>
</body>
</html>
```

从上面的代码中可以看到，我们定义了一个变量 url，并通过代码<a v-bind:href="this.url" v-on:click='validate($event)'> 点我确定做了以下两件事情：

（1）把 url 绑定到了该元素上。

（2）该元素在触发 click 事件时会调用 validate 方法。validate 方法传递了一个特殊的参数 $event，该参数是当前事件的一个实例（MouseEvent）。

在 validate 方法中是这样定义的：先验证是否符合规则，若符合，则放行，然后继续触发 <a/> 元素的默认动作（让浏览器发生跳转）；否则会弹出一个 "alert" 提示框。

使用浏览器打开，可以看到页面如图 6-16 所示改变按钮的默认逻辑，触发表单验证。

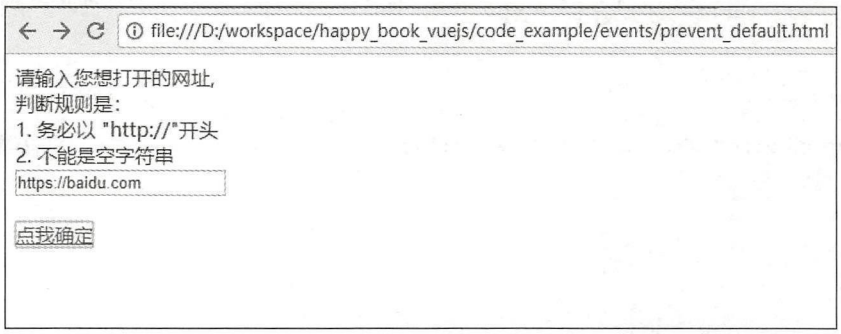


图 6-16　改变按钮的默认逻辑，触发表单验证

先输入一个合法的地址：http://baidu.com，单击后页面发生了跳转，跳转到了百度。再输入一个不合法的地址：https://baidu.com，该地址不是以"http://"开头，所以 Vue.js 代码不会放行。如图 6-17 所示验证不符合规则提示。

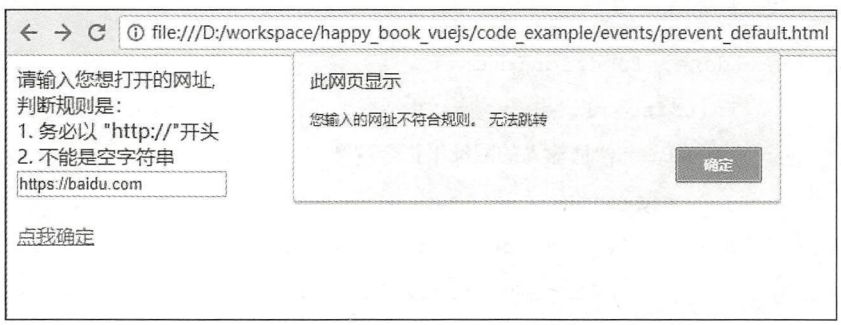


图 6-17　验证不符合规则提示

5. Event Modifiers 事件修饰语

有时我们希望把代码写的优雅一些，但使用传统的方式可能会把代码写的很“臃肿”。如果某个元素在不同的 event 下有不同的表现，那么代码看起来会有很多个 if ...else ...分支。因此，Vue.js 提供了 Event Modifiers。例如，可以把上面的例子略加修改：

```
<html>
<head>
```

```
    <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
</head>
<body>
    <div id='app'>

        请输入您想打开的网址,       <br/>
        判断规则是:                  <br/>
        1. 务必以 "http://"开头      <br/>
        2. 不能是空字符串            <br/>
        <input v-model="url" placeholder="请输入 http:// 开头的字符串, 否则不会跳转"
/> <br/>
        <br/>
        <a v-bind:href="this.url" v-on:click='validate($event)' v-
on:click.prevent='show_message'> 点我确定 </a>
    </div>

    <script>
        var app = new Vue({
            el: '#app',
            data: {
                url: ''
            },
            methods: {
                validate: function(event){
                    if(this.url.length == 0 || this.url.indexOf('http://') != 0){
                        if(event){
                            event.preventDefault()
                        }
                    }
                },
                show_message: function(){
                    alert("您输入的网址不符合规则, 无法跳转")
                }
            }
        })
    </script>
```

```
</body>
</html>
```

可以看出上面的代码核心是：

```
<a v-bind:href="this.url" v-on:click='validate($event)' v-
on:click.prevent='show_message'> 点我确定 </a>

methods: {
    validate: function(event){
        if(this.url.length == 0 || this.url.indexOf('http://') != 0){
            if(event){
                event.preventDefault()
            }
        }
    },
    show_message: function(){
        alert("您输入的网址不符合规则，无法跳转")
    }
}
```

首先在<a/>中定义了两个 click 事件：click 和 click.prevent，后者表示如果该元素的 click 事件被阻止了，应该触发什么动作。然后在 methods 代码段中专门定义了 show_message，用于给 click.prevent 使用。上面的代码运行与前一个例子是一样的，只是抽象分类的程度更高一些，在复杂项目中有用。

这样的 Event Modifiers 有以下几种。

- stop propagation 停止（调用了 event.stopPropagation()方法）后，被触发。
- prevent 调用了 event.preventDefault()后被触发。
- capture 子元素中的事件可以在该元素中被触发。
- self 事件的 event.target 是本元素时被触发。
- once 事件最多被触发一次。
- passive 为移动设备使用（在 addEventListeners 定义时增加 passive 选项）。

以上的 Event Modifiers 也可以连接起来使用，如 v-on:click.prevent.self。

6. Key Modifiers 按键修饰语

Vue.js 很贴心地提供了 Key Modifiers，也就是一种支持键盘事件的快捷方法。看下面的例子：

```
<html>
<head>
    <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
</head>
<body>
    <div id='app'>
        输入完毕后，按下回车键，就会<br/>
        触发 "show_message" 事件~  <br/><br/>

        <input v-on:keyup.enter="show_message" v-model="message" />
    </div>

    <script>
        var app = new Vue({
            el: '#app',
            data: {
                message: ''
            },
            methods: {
                show_message: function(){
                    alert("您输入了: " + this.message)
                }
            }
        })
    </script>
</body>
</html>
```

在上面的代码中，v-on:keyup.enter="show_message" 为 <a/> 元素定义了事件，该事件对应回车键（严格地说，是回车键被按下后松开弹起来的那一刻）。

使用浏览器打开上面代码对应的文件，输入一段文字并按回车键后，就可以看到事件已经被触发了。如图 6-18 所示使用 Key Modifiers 触发事件。

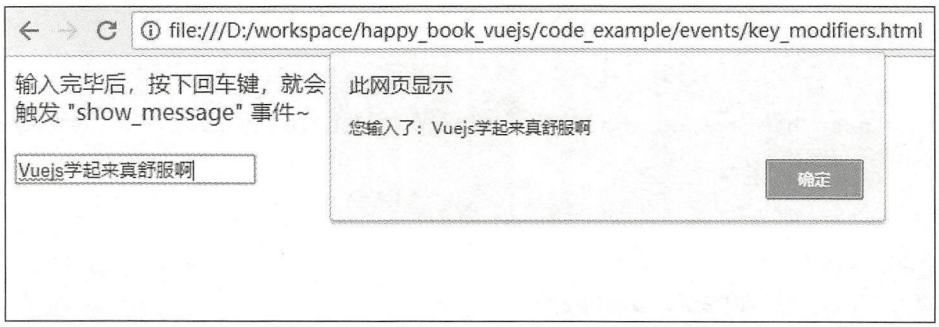


图 6-18　使用 Key Modifiers 触发事件

Vue.js 支持以下 Key Modifiers。

- Enter 回车键
- tab Tab 键
- delete 同时对应 Backspace 和 Delete 键
- esc ESC 键
- space 空格键
- up 向上键
- down 向下键
- left 向左键
- right 向右键

随着 Vue.js 版本的不断迭代和更新，越来越多的 Key modifiers 被添加了进来，如 page down,ctrl。对于这些键的用法，大家可以查阅官方文档。

6.10 与 CSS 预处理器结合使用

《程序员修炼之道》这本书中曾提到程序员的一个职业习惯——DRY（Don't Repeat Yourself），即不要做重复的事情。

目前的编程语言几乎都具备了消灭重复代码的能力，但是 CSS 是唯一不具备支持变量的编程语言。因为 CSS 本身只是一个 DSL（Domain Specific Language，领域特定的语言），不是“编程语言”。这样也就决定了它的特点 ：上手快，可以很好地表现 HTML 中某个元素的外观。缺点就是无法通过常见的重构手法（Extract Method，Extract Variable 等）精简代码。

因此，SCSS、SASS、LESS 等一系列的“CSS 预处理器”（precompiler）应运而生。

6.10.1 SCSS

SCSS 的全称为 Sassy CSS（时髦的 CSS），是 SASS 3 引入的新语法，其语法完全兼容

CSS 3，并且继承了 SASS 的强大功能。也就是说，任何标准的 CSS 3 样式表是具有相同语义的、有效的 SCSS 文件。官方网站同 SASS。

由于 SCSS 是 CSS 的扩展，因此所有在 CSS 中正常工作的代码也能在 SCSS 中正常工作。也就是说，对于一个 SASS 用户，只需要理解 SASS 扩展部分如何工作的，就能完全理解 SCSS。

大部分的用法都与 SASS 相同。 唯一不同的是，SCSS 需要使用分号和大括号。SCSS 可以说是全面取代了 SASS。

看下面的例子：

```
$font-stack:    Helvetica, sans-serif;
$primary-color: #333;

body {
  font: 100% $font-stack;
  color: $primary-color;
}
```

在上面的代码中，定义了两个变量：$font-stack 和$primary-color。编译后的 CSS 如下：

```
body {
  font: 100% Helvetica, sans-serif;
  color: #333;
}
```

更多内容可以到官方网站进行学习，网址为 https://sass-lang.com/guide。

6.10.2　LESS

LESS 也是一种 CSS 预处理器。它是只多了“一丢丢”内容的 CSS（It's CSS, with just a little more）。

LESS 的官方网址为 http://lesscss.org/，github 的官方网址为 https://github.com/less/less.js。其作用与 SCSS 一样，也是为了让代码更加精简，删除无意义的重复代码。我们来看下面的例子：

```
// Variables
@link-color:       #428bca; // sea blue
@link-color-hover:  darken(@link-color, 10%);

// Usage
a,
.link {
```

```
  color: @link-color;
}
a:hover {
  color: @link-color-hover;
}
.widget {
  color: #fff;
  background: @link-color;
}
```

上面的例子中定义了两个变量：@link-color 和 @link-color-hover，并且在下方进行了引用。同时还使用了换算功能 darken(@link-color, 10%)。

上面的代码会被编译成下面的 CSS：

```
a,
.link {
  color: #428bca;
}
a:hover {
  color: #3071a9;
}
.widget {
  color: #fff;
  background: #428bca;
}
```

可以看到，LESS 的功能非常强大。

6.10.3 SASS

提到 SCSS、LESS，就不得不提 SASS。SASS 的官方网址为：https://sass-lang.com/，github 的官方网址为 https://github.com/sass/sass。

SASS 的特点是去掉了大括号和分号，看起来特别简单，使用空格来标记不同的段落层次。与 HAML 基本是一样的。

我们来看下面的例子：

```
$font-stack:    Helvetica, sans-serif
$primary-color: #333
```

```
body
  font: 100% $font-stack
  color: $primary-color
```

在上面的代码中定义了两个变量：$font-stack 和 $primary-color，并且在下面对它们进行了引用。

我们来看编译后的结果：

```
body {
  font: 100% Helvetica, sans-serif;
  color: #333;
}
```

不过，在实际应用中却很少使用该语言。因为在实际应用中，程序员喜欢把 UI 或美工或前端工程师给过来的 CSS 文件直接使用。如果使用 SASS 的话，就很尴尬，还需要再动手做一遍转换，比较浪费时间。而且虽然美工可以看懂 CSS，但是看不懂 SASS。因此，这个技术比较落后，慢慢地被 SCSS（SASS 3.0）取代。

6.10.4　在 Vue.js 中使用 CSS 预编译器

CSS 预编译器使用的前提是我们以 Webpack 的形式使用 Vue.js。这里以 SASS 为例：

安装依赖 "sass-loader" 和 "node-sass"。运行下面的命令：

```
$ npm i sass-loader node-sass -D
```

在"webpack.base.conf.js"中添加相关配置。

```
{
   test: /\.s[a|c]ss$/,
   loader: 'style!css!sass'
}
```

在对应的 ".vue" 文件中，可以这样定义某个样式：

```
<style lang='sass'>
td {
  border-bottom: 1px solid grey;
}
</style>
```

上面的代码在运行时，会被 Webpack 编译成对应的 CSS 文件。

6.11 自定义 Directive

Vue.js 除了自身提供的 v-if、v-model 等标准的 Directive 外，还提供了非常强大的自定义功能。使用这个功能，可以定义属于自己的 Directive。

6.11.1 例子

我们来看下面的例子：

```
<html>
<head>
    <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
</head>
<body>
    <div id='app'>
        下面是使用了自定义 Directive 的 input，可以自动聚焦（调用 focus() 方法）：<br/>
        <br/>
        <input v-myinput/>
    </div>
    <script>
        var app = new Vue({
            el: '#app',
            directives: {
                "myinput": {
                    inserted: function(element){
                        element.focus()
                    }
                }
            }
        })
    </script>
</body>
</html>
```

上面的代码中，先在 Vue 中定义了一个 directives 代码段。

```
directives: {
```

```
    myinput: {
        inserted: function(element){
            element.focus()
        }
    }
}
```

- myinput：自定义 Directive 的名字。使用时就是 v-myinput。
- inserted：这是一个定义好的方法（钩子方法），表示页面被 Vue.js 渲染的过程中，在该 DOM 被 "insert"（插入）到页面时被触发，内容是 element.focus()。

使用浏览器打开后，可以看到<input/> 标签是会自动聚焦的。此时，用户就可以直接输入内容了，如图 6-19 所示的自定义标签实现自动聚焦。

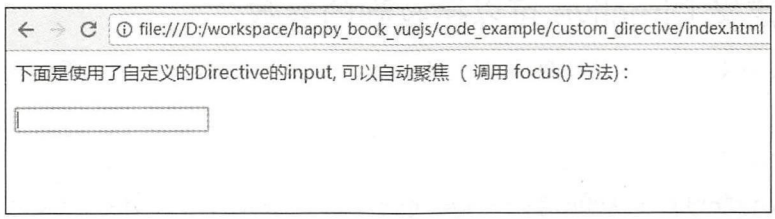


图 6-19　自定义标签实现自动聚焦

如果在 "Webpack" 环境下，也可以使用这样的方法：

```
Vue.directive('myinput', {
  inserted: function (elemenet) {
    element.focus()
  }
})
```

6.11.2　自定义 Directive 的命名方法

如果希望把 v-myinput 的调用写成 v-my-input，在定义时，就应该：

```
directives: {
    // 注意下面的写法. 使用双引号括起来
    "my-input": {
        inserted: function(element){
            element.focus()
        }
    }
}
```

这样就可以在 View 中使用了。

```
<input v-my-input />
```

6.11.3 钩子方法（Hook Functions）

我们在上面的例子中知道了 inserted 是一个钩子方法。下面是一个完整的列表：

- bind：只运行一次，当该元素首次被渲染时（绑定到页面时）。
- inserted：该元素被插入到父节点时（也可以认为是该元素被 Vue 渲染时）。
- update：该元素被更新时。
- componentUpdated：包含的 component 被更新时。
- unbind：只运行一次，当该元素被 Vue.js 从页面解除绑定时。

6.11.4 自定义 Directive 可以接收到的参数

Vue.js 为自定义 Directive 实现了强大的功能，可以接收多个参数。看下面的例子：

```
<html>
<head>
    <script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>
</head>
<body>
    <div id='app'>
        下面是一个非常全面的、自定义 Directive 的例子： <br/>
        <br/>
        <input v-my-input:foo.click="say_hi" />
    </div>
    <script>
        var a
        var app = new Vue({
            el: '#app',
            data: {
                say_hi: '你好啊，我是个 value'
            },
            directives: {
                "my-input": {
                    inserted: function(element, binding, vnode){
                        element.focus()
                        console.info("binding.name: " + binding.name)
                        console.info("binding.value: " + binding.value)
                        console.info("binding.expression: " + binding.expression)
                        console.info("binding.argument: " + binding.arg)
                        console.info("binding.modifiers: ")
```

```
                    console.info(binding.modifiers)
                    console.info("vnode keys:")
                    console.info(vnode)
                }
            }
        }
    })
</script>
</body>
</html>
```

如图 6-20 所示的自定义 Directive 接收的参数打印结果。

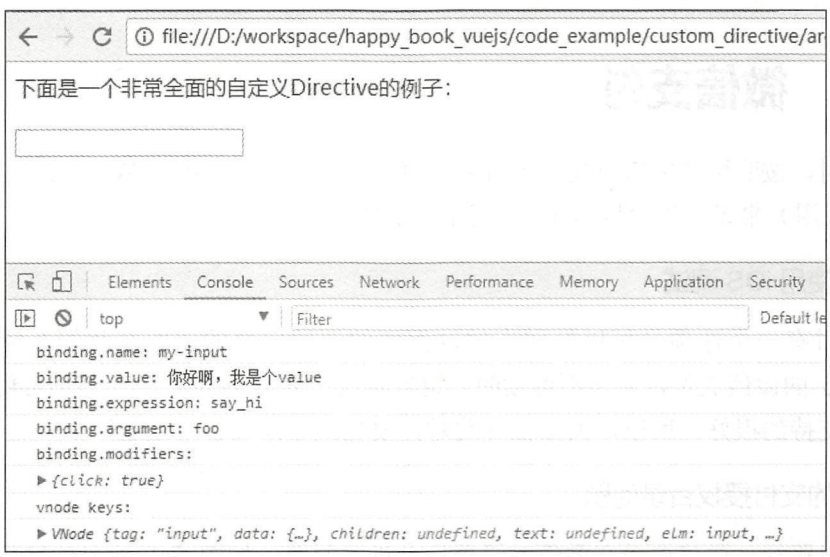


图 6-20　自定义 Directive 接收的参数打印

从图 6-20 中，可以看出，自定义 Directive 在声明时，接收了三个参数：function（element、binding、vnode）。通过这三个参数，就可以看到很多对应的内容，包括 binding.name、binding.value 和 binding.expression，它们的含义都是字面上的意思。借助这些内容，可以实现自己想要的 Directive。

6.11.5　实战经验

（1）优先考虑使用 Component。

考虑到维护成本，其作用与 JSP 中的自定义标签是一样的。与其使用 Directive，不如使用 Component。

（2）如果一定要用，就把它实现的尽量简单。

如果接手的新人水平太浅，那么很可能读不懂这段代码。

第 7 章

实战周边及相关工具

本章的几个问题都曾经耗费笔者几天到几周的时间，这里一并列举出来。前人踩坑，后人绕路，希望我们的经验对读者有用。

7.1 微信支付

微信支付，按照微信的官方文档来看并不算难，特别是“传统的 Web 项目”。但是对于 SPA（单页应用）来说就很“坑”了，几乎没有解释。

1. 优先使用 iOS 调试

微信支付有一个选项是可以打印支付过程中的调试信息的。但是我们在使用过程中发现， Android 的微信支付错误是不可读的。也就是说，开启 debug 选项是不可用的。而对于苹果设备就支持的很好。所以，大家在开发时，要先把苹果设备“走通”。

2. 微信的支付授权目录问题

对于支付路径，微信要求在微信的管理后台进行配置。如图 7-1 所示的微信支付的支付授权目录设置。

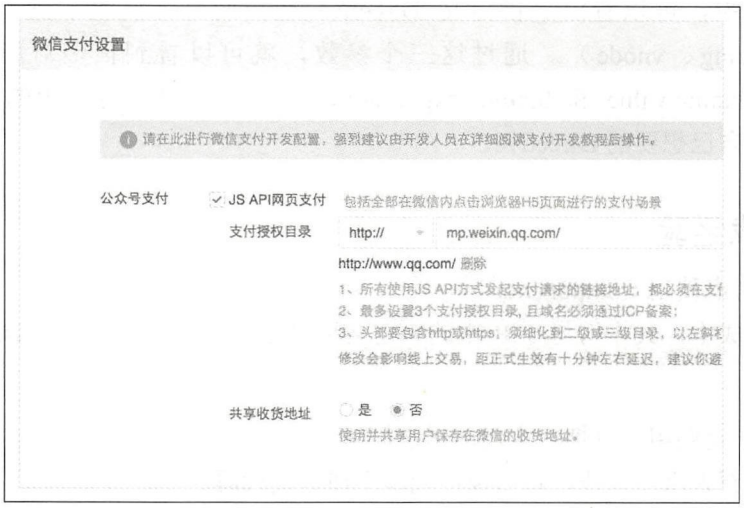


图 7-1　微信支付的支付授权目录设置

注意，Android 和 iOS 的配置是不一样的。Android 取支付页面的 URL，iOS 取根路径 URL。例如，根路径是 http://yoursite.com，支付路径是 http://yoursite.com/#/books/pay?id=3。那么，在设置“支付授权目录”时，需要设置以下两个目录。

- http://yoursite.com/#/（给 iOS）；
- http://yoursite.com/#/books/（给 Android）。

7.2 Hybrid App：混合式 App

目前 App 几乎是每个互联网公司的标配，但不是每个团队都具备开发原生 App 的能力。于是就出现了以 Phonegap、Titanium、Xamarin、React Native 等一系列的混合式 App，我们大概了解一下。

- Phonegap：出现的比较早，使用了很多 HTML5 的技术实现原生 App 的功能，如拍照等。不过完全没有实用价值，响应速度非常慢，卡顿明显。
- Titanium：出现的比较早，性能很好，与小程序很类似。使用 js、css 的类似技术，在不同平台上只要稍加修改代码，就可以跨平台媲美原生 App。缺点是有学习曲线，特别是 module 很难写。公司已被收购。
- Xamarin：使用.NET 实现，与 titanium 很接近。也有不少的使用群体。不过由于是微软的平台，属于闭源技术，所以不具备开源社区的支持。
- React Native：使用 js 黑科技，直接生成 Android/iOS，原理与 Titanium、Xamarin 一样。目前比较流行的是混合式开发方式，性能也很高。

1. 共同的缺点

无论是 Titanium、Xamarin 还是 React Native、Weex，都属于使用 js/.net 把代码改造成可以被 Android/iOS 的 Javascript Virtual Machine 所能接受的情况。也就是说，对原生的 Android/iOS 平台做了封装。这样做的好处是为两个不同的编程语言增加了一个统一的编程入口。

缺点也非常明显：做一些简单的事情（展示页面、单击按钮等）没问题，一旦需要用到第三方应用（定制化的地图、定制化的身份证识别、人脸识别等），就要求开发人员具备骗写“native module”的能力。开发人员必须同时精通 Android / iOS（编写这种 native module 比单纯使用的要求高很多），但这样做背离了这些技术的初衷（让不太懂 App 的编程人员可以快速上手开发）。

2. 原生的壳儿 + Webview 的开发方式

还有一种就是原生的壳儿 + Webview 的开发方式。

- 原生的壳儿是指外壳部分完全使用 100%的 native App。

- Webview 是指所有页面都放到 Webview 中展示。

经过实践证明，这种情况对于 iOS 设备是可行的，但 Android 设备是不可行的。也就是说，原生的壳儿 + Webview 的开发方式只适用于 iOS。

（1）iOS：机器硬件性能好，因为软件使用 Object C 开发，所以使用起来效果非常棒，与 native App 的体验是一样的。每个页面都可以瞬间打开，并且页面滑动非常流畅。在开发层面上几乎不用考虑页面的适配，解决了很大的问题。

（2）Android：机器硬件性能稍差于苹果，软件使用 Java 开发，性能比 Object C 差。在 Android 中，Webview 的性能体验比 iOS 的差太多，并且每个页面打开速度是 3 ~ 10 秒，卡顿严重。

我们曾经试着在优化的路上走了一段时间，发现对于 Android + Webview（Vue.js）的方式，前期开发成本低于原生，后期维护成本远高于原生，而且一些问题在混合架构下基本无解。

因此，我们得出的结论是：开发 App、iOS 可以使用混合式开发，Android 务必使用原生开发。

7.3 安装 Vue.js 的开发工具： Vue.js devtool

Vue.js 是一个框架，构建于 JavaScript 的代码之上，而 JavaScript 语言的实现并不像其他后端语言（如 Java、Python、Ruby 等）那样有着对 debug 友好的错误提示机制。原因如下：

（1）JavaScript 是一种非常灵活的语言，支持“元编程”，而对于任何一个“框架”来说，都会大量用到“元编程”的能力。例如：

```
var my_code = "var a = 1 + 1; console.info(a) "
eval(some_code)
```

在上面的代码中，eval 就是一个典型的“元编程”方法（meta programming)。通俗来讲，“元编程”就是为了“让程序来写程序”。“元编程”能力是评估一种语言是否“高级”的一个重要指标。

“元编程”的缺点是显示错误信息比较麻烦，往往显示的错误提示或 stack trace 不是特别明朗。

（2）JavaScript 语言是在各种不同的浏览器中实现的。因为不同的浏览器厂家都会实现不同的 JavaScript 虚拟机（ virtual machine ），所以我们会发现一些 JavaScript 的错误往往是不可读的，如图 7-2 所示。

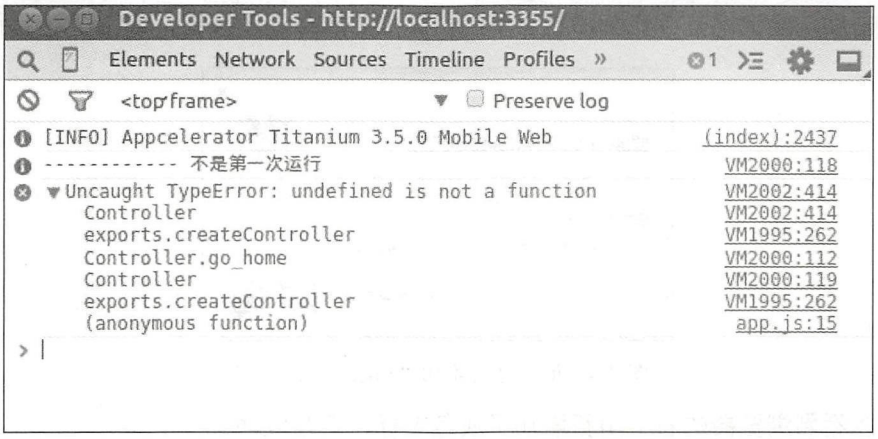


图 7-5　JavaScript 的错误不可读

上面出错的原因，就是由于 JavaScript 的代码中可能会有不同的"scope"，而该浏览器为每个"scope"都会分配一个 "virtual machine"，因此上面打印出来的 stack trace 很忠实地反应出了问题的所在，就是几乎没有可读性。

为了方便开发，读者一定要安装对应的开发组件，如"Vue.js devtools"。

官方网址为 https://github.com/Vue.js/vue-devtools。

1. 安装步骤

安装非常简单，建议读者使用 chrome，这样就可以以插件的形式进行安装了。

步骤 01 安装 chrome。

步骤 02 打开 https://chrome.google.com/webstore/detail/Vue.js-devtools/ nhdogjmejiglipccpnnnanhbledajbpd。

步骤 03 可以看到如图 7-3 所示的 Vue.js devtools 的插件主页。

图 7-3　Vue.js devtools 的插件主页

步骤 04 单击后，会询问是否安装，这里单击“添加扩展程序”按钮，如图 7-4 所示。

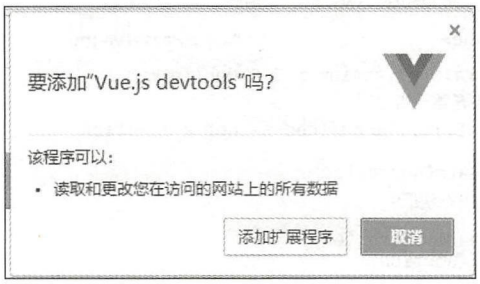


图 7-4　询问是否添加“Vue.js devtools”

步骤 05 可以看到浏览器的右上角新增加了灰色图标，安装成功。

步骤 06 执行“设置”→“更多工具”→“扩展程序”命令，就可以看到刚才安装的 Vue.js devtools 了，选中“允许访问文件网址”复选框，如图 7-5 所示。

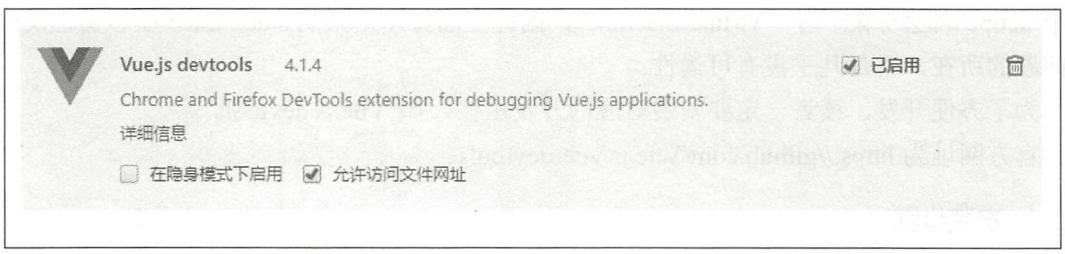


图 7-5　选中“允许访问文件网址”复选框

2. 使用步骤

在安装 Vue.js devtools 之前，我们调试的时候，都是打开浏览器的 Developer Tools，如图 7-6 所示。

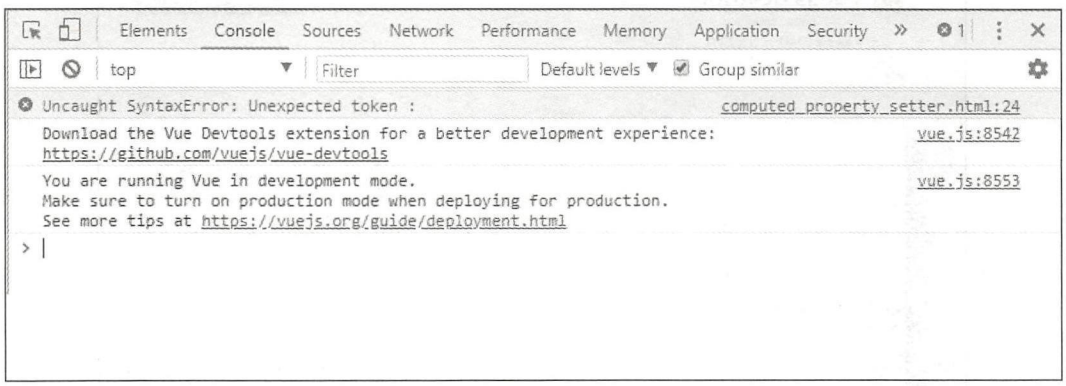


图 7-6　打开浏览器的 Developer Tools

安装好之后，如果 Vue.js 项目是使用 HTTP 服务器打开（不是 file:///...）的，就可以看到如图 7-7 所示的界面。

图 7-7　Vue.js devtools 的主界面

7.4 如何阅读官方文档

我们在查看 vue-js 文档的时候，会发现它与真正使用的项目代码完全不一样。例如，Vue.js 官方文档的讲解是完全把所有代码写在 js 中的：

```
var Child = {
  template: '
A custom component!
'
}
new Vue({
  // ...
  components: {
    //  将只在父模板可用
    'my-component': Child
  }
})
```

而实际的项目代码是这样的：

```
<template>
....
</template>
<script>
.....
</script>
<style>
```

```
</style>
```

原因就在于，我们在实际项目中使用了 vue-loader，可以非常好的自动加载所有内容，按照 Webpacke+vue 的约定。

1. webpack 官方文档地址

webpack 就是一种工具，可以把各种 js/css/html 代码打包编译到一起。

Vue.js 中已经集成了这个工具，我们在使用 vue-cli 时就会根据命令来生成 webpack 所要求的文件结构，然后在打包时（npm run build），Vue.js 的源代码就会被 webpack 打包成正常的 html 代码和文件目录。

读者要知道在本书中所讲的知识都是"基于 webpack 的 Vue.js"，否则只看 Vue.js 的官方文档是看不懂的。

官方网址为 https://cn.Vue.js.org/，单击右上角的“生态系统”菜单就可以看到入口了。

2. 如何查看 API 文档

对于初学者来说，需要适应 Vue.js 的 API 查看方式。

想要读懂 API，首先需要对 Vue.js 的各个方面有个明确的认识。

- 英文版网址：https://Vue.js.org/v2/api/。
- 中文版网址：https://cn.Vue.js.org/v2/api/。

建议在查看 API 文档时使用英文版，因为很多专业的名词，如 minxin、component 等都是原汁原味的，读起来更加明白一些。

第 8 章

◀ 实战项目 ▶

通过前面的学习，相信大家已经对 Vue.js 有了一个非常全面的了解。

下面通过一个真实的项目来完成 Vue.js 实战。

假设我们在一家互联网电子商务公司就职，该公司的业务是帮助大山里的穷苦农民，把自家的农产品卖到城市。

需要解决的问题有以下三个：

（1）让农民把大山里的东西卖掉。

（2）让都市中的人享受到纯原生态的绿色食品，并且享受更低的价格。

（3）去掉中间商。保证农民的收入更多，消费者消费的价格更低。

通过这样的公益项目，公司也可以解决自身的生存问题。

8.1 准备 1：文字需求

梳理需求是项目的重中之重，把老板的“一句话需求”梳理成条理清晰符合逻辑的文字，再进一步的整理成原型图。

1. 参与的角色

总共是以下三个角色：

- 大山中的农民：提供农产品。
- 城市消费者：来购买农产品。
- 平台管理员：对平台进行日常运作。

参与的角色如图 8-1 所示。

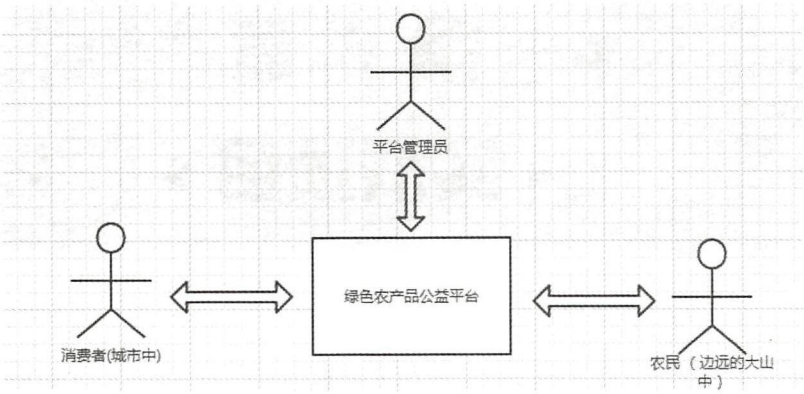


图 8-1　参与的角色

（1）消费者

- 可以注册。
- 可以微信授权。
- 可以查看商品列表页。
- 可以查看商品详情页。
- 可以查看购物车。
- 可以支付商品。

消费者用例图如图 8-2 所示。

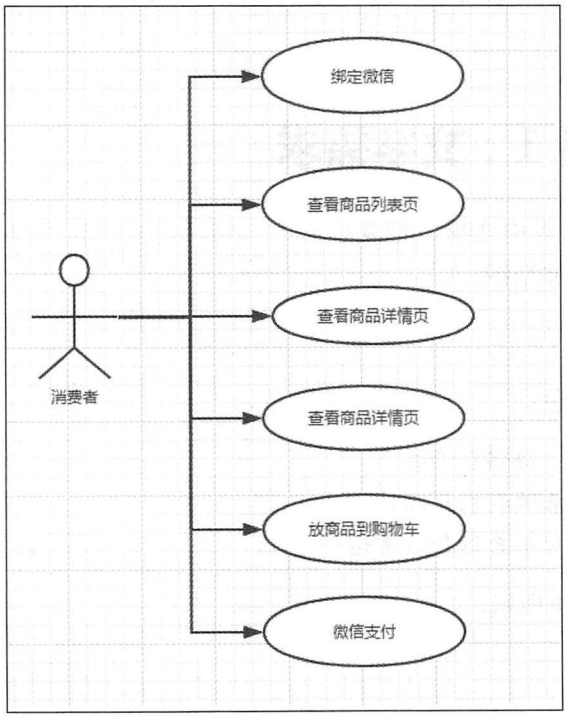


图 8-2　消费者用例图

（2）农民

直接与公司联系，告知可以出售的特产、价格等信息。

农民用例图如图 8-3 所示。

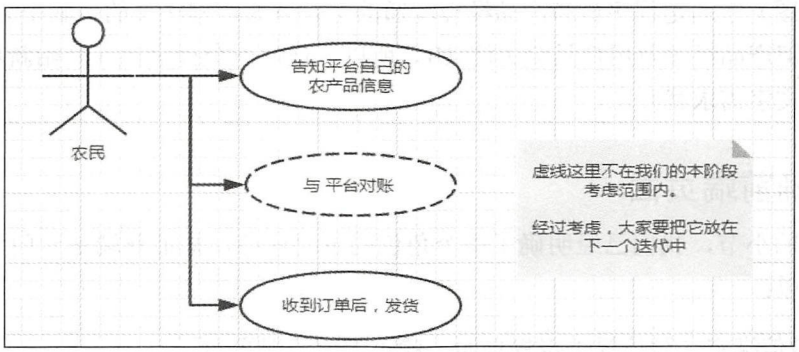


图 8-3　农民用例图

（3）平台管理员

- 可以管理商品分类。
- 可以管理商品的上/下架。
- 可以处理订单。
- 订单付款确认后，线下联系发送快递。

平台管理员用例图如图 8-4 所示。

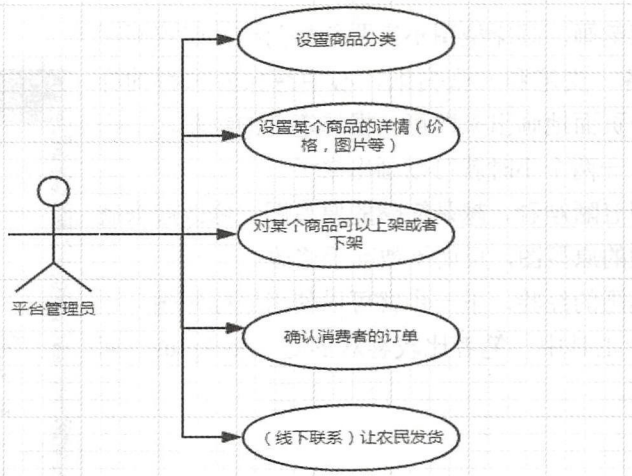
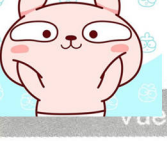


图 8-4　平台管理员用例图



8.2 准备 2：需求原型图

UI（原型图）永远是程序员和产品经理沟通的主要方式。程序员的头脑中会同时关注 UI 和技术实现，但产品经理及用户只会关注 UI。所以，任何一个程序员在开始新项目时，都不能贸然地根据文字需求就开工。

8.2.1 明确前端页面

根据前面的小节，我们已经明确了每个角色的主要任务，知道前端是专门为消费者使用的。消费者可以：

（1）做微信绑定（微信提供授权页面，就不需要注册页面了）。
（2）看到首页。
（3）看到商品列表页。
（4）看到商品详情页。
（5）看到购物车页面。
（6）看到个人信息页面。
（7）看到微信支付页面。

8.2.2 如何画原型图

原型图就是简笔画。画原型图不需要任何门槛，建议新手直接动笔画：准备一支笔和一张白纸，心中想象着项目的样子，一个页面一个页面地画出来即可。根据笔者的经验，一个不太复杂的 APP，一两个小时就可以画出来了。

不要怕原型图丑陋难看。越是简陋的原型图，修改起来就越容易。画得精细的原型图，反而不敢动手修改。

一旦有了动笔画的经验，下一步就可以使用鼠标来画。市面上的原型图设计工具中，笔者比较喜欢的是 Mockplus，简单好用，没有门槛。

图 8-5 首页的原型图

8.2.3 首页

用户打开链接后，直接进入首页，如图 8-5 所示。

在首页中：

- 上部分是轮播图；
- 中间部分是商品分类；
- 下方是商品列表；

- 最下面是 4 个标签页，即首页、商品、购物车、我的。

8.2.4　商品列表页

用户在首页点击商品，即可进入商品列表页，如图 8-6 所示。

图 8-6　商品列表页

8.2.5　商品详情页

用户在商品列表页点击某个商品后，就会进入商品详情页面，如图 8-7 所示。

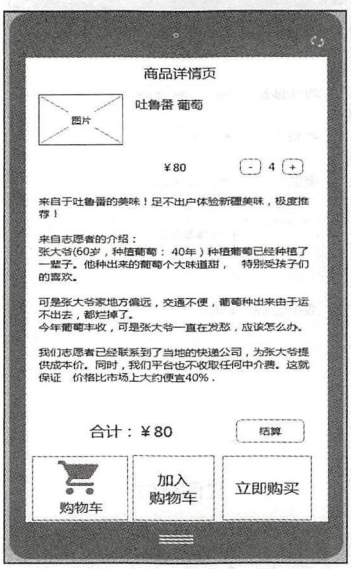


图 8-7　商品详情页

在该页面中，可以看到商品的图文介绍，可以修改购买的数量，也可以直接下单付款。

8.2.6 购物车页面

消费者可以在查看商品时把商品放到购物车中，然后统一结算，如图 8-8 所示。

图 8-8 购物车页面

8.2.7 支付页面

用户可以在购物车中进行支付，也可以在商品购买页中进行支付，如图 8-9 所示。

图 8-9 支付页面

在支付页面会需要显示商品的各种信息、待付金额、用户的收货地址等。确定全部信息无误后，即可进入微信支付页面。

8.2.8　我的页面

用户在首页点击我的，即可进入我的页面，如图 8-10 所示，可以看到自己的头像、微信昵称及历史下单记录。

图 8-10　我的页面

8.2.9　我的订单列表页面

用户在我的页面中点击我的订单，即可进入我的订单页面，如图 8-11 所示。

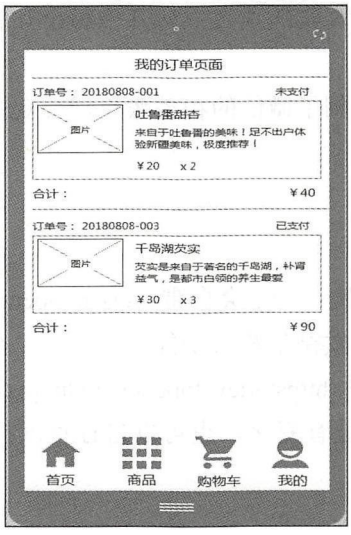


图 8-11　我的订单页面

可以看到历史订单、每个订单的编号、内容和支付状态等信息。

8.2.10 总结

这些原型图直接勾勒出我们要做的项目。

8.3 准备 3：微信的相关账号和开发者工具

8.3.1 微信相关账号的申请

因为微信的 H5 页面会涉及一些功能（登录、分享等），所以需要事先了解一下微信的产品家族。

- 微信公众平台：包括服务号、订阅号。网址为 https://mp.weixin.qq.com。
- 微信开放平台：为“手机 App”提供登录分享等操作。网址为 https://open.weixin.qq.com/。
- 微信商户平台：提供微信支付功能。网址为 https://pay.weixin.qq.com。

无论是新手还是高手，在这里几乎都会发蒙，所以读者务必做好笔记。另外，每申请一个账号，就要把用户名和密码记下来。以上三个平台的账号都是独立的，并且每个账号中有自己的 appid、appkey、app secret 等各种机密的秘钥。一定要妥善保管好，不能混淆，否则会为调试带来很大的困扰。

对于公司来说，需要准备好相关的证照，并且在对应的时间内使用公司账号打款给微信。由于过程比较烦琐，因此这里将申请的步骤省略。

读者不要打退堂鼓，我们在真实的项目中，几乎每个互联网公司都会用到微信公众号的功能。

下面我们假设已经成功申请到了微信的相关账户，公众平台上的是“服务号”且具备支付功能。

8.3.2 微信开发者工具

由于微信自带浏览器的特殊性（一定会自带 weixin openid 等微信独有的信息），会导致我们平时使用的普通浏览器在开发微信相关功能时（授权、分享、支付等）无法使用，因此需要下载微信开发者工具，地址为 https://developers.weixin.qq.com/miniprogram/dev/devtools/download.html。如果网址有变化，也可以自行百度搜索“微信开发者工具”。下载页面如图 8-12 所示。

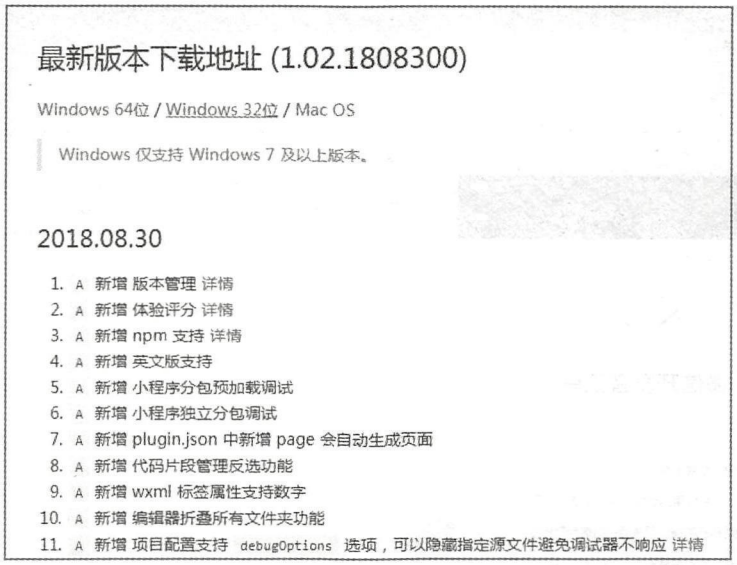


图 8-12　下载页面

下载完成后，双击开始安装。安装后会出现登录页面，如图 8-13 所示。

扫描二维码后，可以看到有两个入口：一个是微信小程序；另一个是公众号网页项目，如图 8-14 所示。

图 8-13　登录页面

图 8-14　两个入口

点击公众号网页项目，可以看到界面几乎与浏览器的开发者工具一样，并且提供了额外的功能。

- 左上角提供了 WIFI 的信号选择。
- 右上角提供了“清缓存”功能。
- 左上角可以看到当前登录的微信用户图标。

如图 8-15 所示。

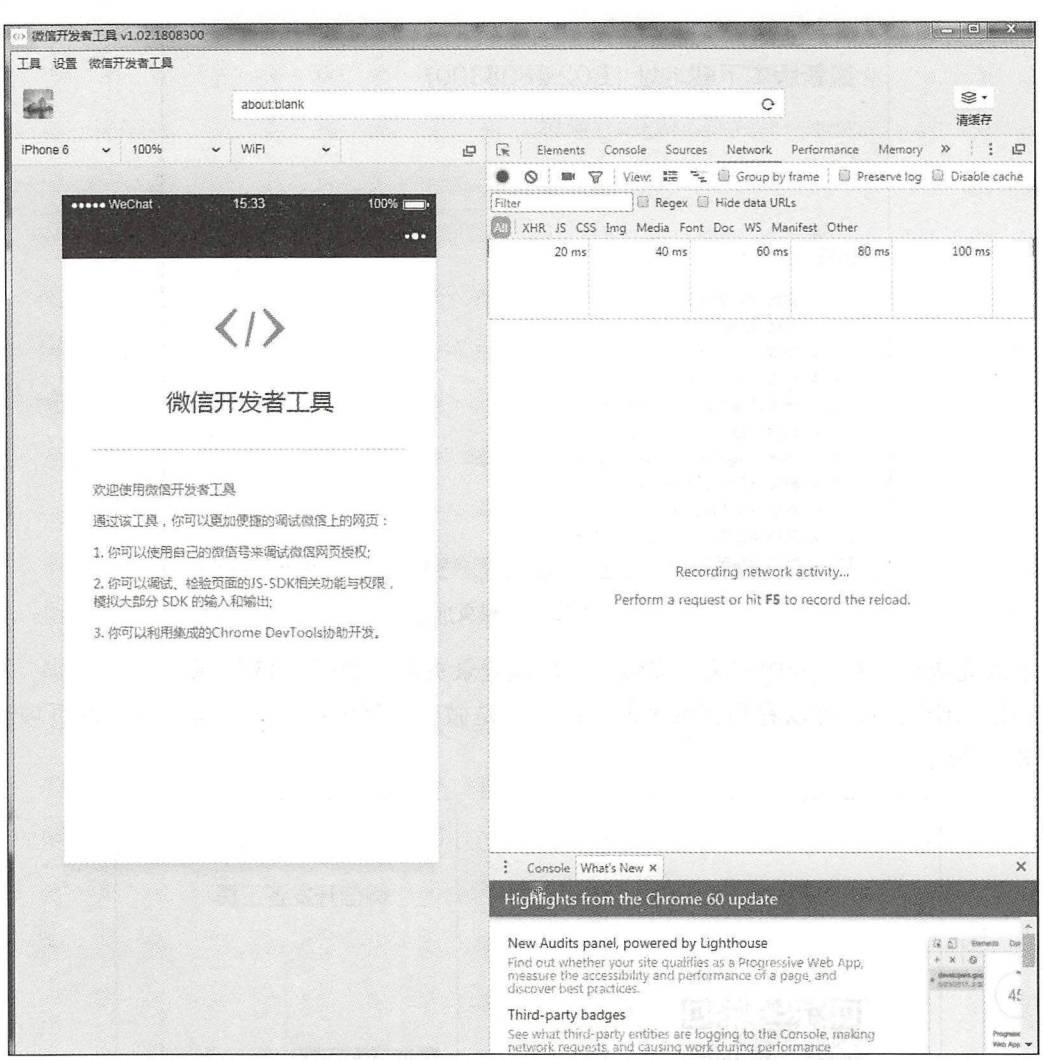


图 8-15　微信开发者工具

建议 Linux 环境开发的读者，暂时回到 Windows 的开发环境，因为 Linux 下没有官方提供的微信开发者工具。

8.4 项目的搭建

创建一个基于 Webpack 模板的新项目：

```
$ vue init webpack shop_h5
```

安装依赖：

```
$ cd shop_h5  && cnpm install
```

在本地以默认端口来运行：

```
$ npm run dev
```

可以看到本地服务器已经运行起来了。在这个阶段，只需要修改它的标题就可以。打开根目录下的 index.html，修改其内容如下：

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>公益爱农</title>
  </head>
  <body>
    <div id="app"></div>
  </body>
</html>
```

使用浏览器打开后，就可以看到一个没有内容的 Vue.js 应用已经运行起来了。

8.5 用户的注册和微信授权

为了追求快速上线，项目组决定去掉传统项目中的用户注册和用户登录页面，直接使用微信授权来实现。

- 用户的微信浏览器会把当前微信用户的 open_id 传递给后台服务器。
- 后台服务器给微信服务器发送请求，获得当前微信用户的信息。
- 后台服务器为该用户生成一个用户文件。
- 后台服务器告知 H5 端已经成功注册该用户。
- H5 端为该用户展示对应的页面。

通过微信授权来注册的过程如图 8-16 所示。

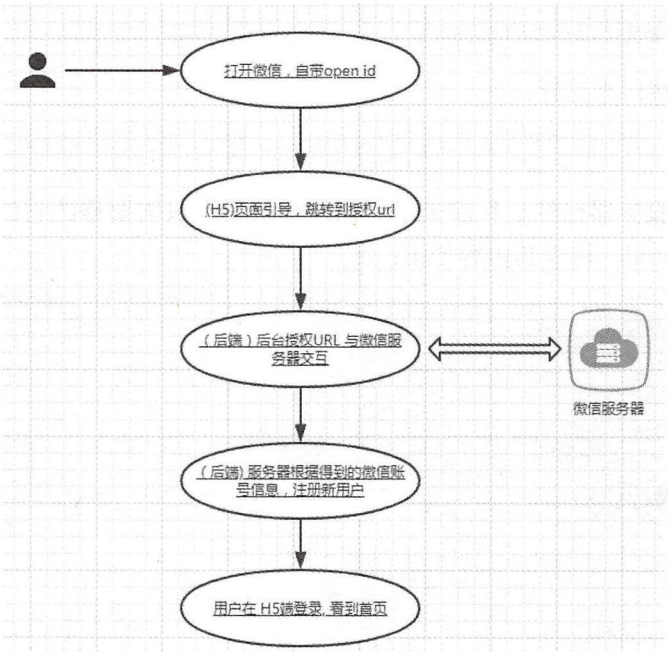


图 8-17　通过微信授权来注册的过程

可以看出，主要代码都是在服务器端实现的。

1. 用户打开首页后直接跳转到后台服务器

（1）修改路由文件 src/router/index.js。

```
Vue.use(Router)

export default new Router({
  routes: [
    {
      path: '/wait_to_shouquan',
      name: 'wait_to_shouquan',
      component: require('../views/wait_to_shouquan.vue')
    },
  ]
})
```

（2）增加 src/views/wait_to_shouquan.vue。

```
<template>
  <div style="padding: 50px;">
```

```
  <h3>正在跳转到授权界面...</h3>
  </div>
</template>

<script>
  export default {
    created () {
      window.location.href = this.$store.state.web_share + "/auth/wechat"
    },
    components: {
    }
  }
</script>
```

上面的代码使用了 Vuex 来保存系统变量。

（3）增加核心模板文件 src/main.vue。

```
<template>
  <div id="app">
    <router-view></router-view>
  </div>
</template>

<script>
import store from './vuex/store'
import { SET_BASEINFO, GET_BASEINFO } from './vuex/mutation_types'
export default {
  store,
  name: 'app',
  data () {
    return {
      user_info: {
        open_id: this.$route.query.open_id
      }
    }
  },
  mounted () {
```

```
    // TODO 开发环境下使用，生产环境下注释
    // store.dispatch(SET_BASEINFO, {open_id: 'opFELv6YkJkMaH-xFkokTWCs5AlQ'})

    if (this.user_info.open_id) {
      store.dispatch(SET_BASEINFO, this.user_info)
    } else {
      store.dispatch(SET_BASEINFO)
      if (store.state.userInfo.open_id === undefined) {
        console.info('用户 id 和 open_id 不存在,跳转到授权等待页面')
        this.$router.push({name: 'wait_to_shouquan'})
      } else {
        console.info('已经有 BASEINFO')
      }
    }
  },
  watch: {
    '$route' (val) {
    }
  },
  methods: {
  },
  components:{
  }
}
</script>
<!-- 下方的 CSS 略过 -->
```

上面代码的 mounted()方法会对当前用户的 open_id 进行判断。如果 open_id 存在，就跳转到首页；如果 open_id 不存在，表示该用户是新用户，就需要跳转到授权等待页面。对应的代码如下：

```
    if (this。user_info。open_id) {
      store.dispatch(SET_BASEINFO, this.user_info)
    } else {
      store.dispatch(SET_BASEINFO)
      if (store.state.userInfo.open_id === undefined) {
```

```
      console.info('用户 id 和 open_id 不存在，  跳转到授权等待页面')
      this.$router.push({name: 'wait_to_shouquan'})
    } else {
      console.info('已经有了 BASEINFO')
    }
  }
```

（4）增加对应的 Vuex 代码。

目前来看，Vuex 需要保存两个信息：用户的 open id 和远程服务器的地址、端口等常量。

（5）增加 src/vuex/store.js，这是最核心的文件。完整代码如下：

```
import Vue from 'vue'
import Vuex from 'vuex'
import userInfo from './modules/user_info'
import tabbar from './modules/tabbar'
import toast from './modules/toast'
import countdown from './modules/countdown'
import products from './modules/products'
import shopping_car from './modules/shopping_car'

import * as actions from './actions'
import * as getters from './getters'

Vue.use(Vuex)
Vue.config.debug = true

const debug = process.env.NODE_ENV !== 'production'

export default new Vuex.Store({
  state: {
    web_share: 'http://shopweb.siwei.me',
    h5_share: 'http://shoph5.siwei.me/?#'
  },
  actions,
  getters,
  modules: {
```

```
    products,
    shopping_car,
    userInfo,
    tabbar,
    toast,
    countdown
  },
  strict: debug,
  middlewares: debug ? [] : []
})
```

在上面的代码中，部分代码是在后面会陆续用到的。不用过多考虑，只需要关注下面几行代码：

```
export default new Vuex.Store({
  // 这里定义了若干系统常量
  state: {
    web_share: 'http://shopweb.siwei.me',
    h5_share: 'http://shoph5.siwei.me/?#'
  },
  modules: {
      // 这里定义了当前用户的各种信息， 我们把它封装成一个 js 对象
    userInfo,
  },

})
```

（6）增加 vuex/modules/user_info.js 文件。该文件定义了用户信息的各种属性，代码如下：

```
import {
  SET_BASEINFO,
  CLEAR_BASEINFO,
  GET_BASEINFO,
  COMMEN_ROLE,
  GET_BGCOLOR,
  GET_FONTCOLOR,
  GET_BORDERCOLOR,
  GET_ACTIVECOLOR,
```

```
  EXCHANGE_ROLE
} from '../mutation_types'

const state = {
  id: undefined, //用户 id
  open_id: undefined, // 用户 open_id
  role: undefined
}

const mutations = {
  //设置用户个人信息
  [SET_BASEINFO] (state, data) {
    try {
      state.id = data.id
      state.open_id = data.open_id
      state.role = data.role
    } catch (err) {
      console.log(err)
    }
  },
  //注销用户操作
  [CLEAR_BASEINFO] (state) {
    console.info('清理缓存')
    window.localStorage.clear()
  },
}

const getters = {
  [GET_BASEINFO]: state => {
    console.info('进入到了 getter 中了')
    let localStorage = window.localStorage
    let user_info
    if (localStorage.getItem('SLLG_BASEINFO')) {
      console.info('有数据')
      user_info = JSON.parse(localStorage.getItem('SLLG_BASEINFO'))
    } else {
```

```
      console.info('没有数据')
    }
    return user_info
  },
  [COMMEN_ROLE]: state => {
    if (state.role === 'yonghu') {
      return true
    } else {
      return false
    }
  },
}



const actions = {
  [SET_BASEINFO] ({ commit, state }, data) {
    //保存信息
    if (data !== undefined) {
      let localStorage = window.localStorage
      localStorage.setItem('BASEINFO', JSON.stringify(data))
      commit(SET_BASEINFO, data)
    } else {
      if (localStorage.getItem('BASEINFO')) {
        data = JSON.parse(localStorage.getItem('BASEINFO'))
        commit(SET_BASEINFO, data)
      } else {
      }
    }
  }
}

export default {
  state,
  mutations,
  actions,
```

```
  getters
}
```

（7）增加 src/vuex/mutation_types.js 文件。该文件定义了对象的两个操作，内容如下：

```
export const SET_BASEINFO = 'SET_BASEINFO'
export const GET_BASEINFO = 'GET_BASEINFO'
```

（8）增加 src/vuex/modules/actions.js 文件。该文件对 mutation_types 进行了引用，内容如下：

```
import * as types from './mutation_types'
```

（9）增加 src/vuex/modules/getters.js 文件，内容为空。

```
// 先放成空内容
```

2. 与后端的对接

后端为我们提供了一个链接：http://shopweb.siwei.me/auth/wechat 作为授权页面。我们让 H5 页面直接跳转到该链接即可，不需要添加任何参数。

3. 效果图

在微信开发者工具中打开页面，发现页面会自动跳转。其实有两次跳转：

第一次跳转到 http://shopweb.siwei.me/auth/wechat；第二次跳转到 https://open.weixin.qq.com/connect/oauth2/authorize.....。

微信授权页面如图 8-18 所示。

图 8-18　微信授权页面

点击“确认登录”按钮，进行授权后，就会进入到 H5 的首页。

4. 总结

本节为了让用户跳转到微信授权页面并自动注册，我们做了如下程序层面的内容。

- 使用 Vuex 记录系统常量（远程服务器的地址）。
- 使用 Vuex 记录用户的信息（新增了一个对象 user_info)。
- 使用了一个独立的页面（等待微信授权页面）。
- 每次打开首页之前，都要判断该用户是否登录。
- （后端）让该用户在微信端授权并自动注册，然后把数据返回给前端。

Vuex 是 Vue.js 既复杂又不好理解的模块，这是由 javascript 的语言特性决定的。

另外，本节对于后端的是个挑战，需要对微信返回的数据结构和微信配置很熟悉。由于本书篇幅所限，所以省略后端内容。

8.6 登录状态的保持

对于经典的 Web 开发，都是把当前用户的信息保存到 session 中，需要的时候读取。

对于前端，从理论上讲有以下两种方式：

- 读取 cookie（适合所有 H5 框架）；
- 读取 vuex（适合 Vue.js）。

1. 简单版：使用 cookie

cookie 是明文存储，可以通过调用 document.cookie 来操作。例如，打开浏览器的 console，输入：

```
document.cookie
```

结果：

```
"TY_SESSION_ID=fad74371-40d1-444b-9d1c-5dd33c086b20;
uuid_tt_dd=10_37220323210-1535781572649-914811;
dc_session_id=10_1535781572649.670168;
Hm_lvt_6bcd52f51e9b3dce32bec4a3997715ac=1535781569,1535873915; dc_tos=pef3wv;
Hm_lpvt_6bcd52f51e9b3dce32bec4a3997715ac=1535873935"
```

里面有哪些内容就一目了然了。所以，从安全性的角度来讲，用户的登录信息容易被人盗走。

2. 使用 Vuex

一旦熟悉了的 Vuex 写法，还是很容易的使用。

另外，Vuex 会对很多信息进行封装和作用域的判断，提高了安全性。

保存信息的代码如下：

```
store.dispatch(SET_BASEINFO, this.user_info)
```

读取信息的代码如下：

```
store.state.userInfo
```

8.7 首页轮播图

用户登录到首页之后，首先注意到的就是轮播图，下面就为首页增加该功能。

1. 增加路由

在路由文件 src/router/index.js 的对应位置增加如下代码。

```
import Index from '@/views/shops/index'

Vue.use(Router)

export default new Router({
  routes: [
    {
      path: '/',
      name: 'Home',
      component: Index
    }
  ]
```

2. 增加 vue 页面

增加 src/views/shops/index.vue，内容如下：

```
<template>
  <div class="background">
    <div class="home">
      <div class="m_layout">
```

```
        <!-- 轮播图-->
        <HomeBannerView></HomeBannerView>
        <!--导航-->
        <HomeNavView></HomeNavView>

        <span class="divider" style="height: 4px;"></span>
        <div class="product_top">
          <div class="product_left">
            <div>商品列表</div>
          </div>
        </div>
        <span class="divider" style="height: 2px;"></span>
        <!-- 特产商品 -->

<SpecialMarket :id="good.id" :name="good.name" :description="good.description"
 :image_url="good.image_url" v-for="good in goods"></SpecialMarket>
      </div>
    </div>
    <NavBottomView :is_shops_index="is_shops_index"></NavBottomView>
  </div>
</template>
<script>
    // 轮播图需要的 js 文件
    import {bindEvent,scrollPic} from '../../libs/index.js'
    // 轮播图需要的前台文件
    import HomeBannerView from '../../components/HomeBanner.vue';
    // 商品分类
    import HomeNavView from '../../components/HomeNav.vue';

    // 特产，商品的列表
    import SpecialMarket from '../../components/SpecialMarket.vue';

    // 底部 4 个 TAB， 下一节会讲到
    import NavBottomView from '../../components/NavBottom.vue';

    export default{
```

```
    data () {
      return {
        goods: [],
        is_shops_index: true,
      }
    },
     components:{
      HomeHeaderView,
      HomeBannerView,
      HomeNavView,
      HomeMainView,
      SpecialMarket,
      NavBottomView
     },
     mounted () {
      //bindEvent();
      scrollPic();
      this.loadPage ();
     },
     computed:  {
     },
     methods: {
       loadPage () {
         this.$http.get(this.$configs.api +
'goods/get_goods').then((response)=>{
           console.info(response.body)
           this.goods= response.body.goods
         },(error) => {
           console.error(error)
         });
       },
     }
   }
</script>
```

上面的代码中，先读取了一个 API，然后渲染数据并调用首页的轮播图。

3. 增加轮播图组件

做开发的核心方法论：不要造轮子。一定要造轮子的话，先搜索是否有现成的。而轮播图是最常见的组件之一，所以一定有其他人写好的第三方包，我们拿来用就好。

（1）增加轮播图的组件。在 src/views/shops/index.vue 中增加：

```
import {bindEvent,scrollPic} from '../../libs/index.js'
```

（2）增加对应的文件 libs/index.js。

```
function $id(id) {
    return document.getElementById(id);
}

function bindEvent() {
    var sea = $id("my_search");
    /*banner 对象*/
    var banner = $id("my_banner");
    /*高度*/
    var height = banner.offsetHeight;
    window.onscroll = function() {
        var top = document.body.scrollTop;
        /*当滚动高度大于 banner 的高度时颜色不变*/
        if (top > height) {
            sea.style.background = "rgba(201,21,35,0.85)";
        } else {
            var op = top / height * 0.85;
            sea.style.background = "rgba(201,21,35," + op + ")";
        }
    };
}

function scrollPic() {
    var imgBox = document.getElementsByClassName("banner_box")[0];
    var width = $id("my_banner").offsetWidth;
    var pointBox = document.getElementsByClassName("point_box")[0];
    var ols = pointBox.children;
    var indexx = 1;
```

```
var timer = null;
var moveX = 0;
var endX = 0;
var startX = 0;
var square = 0;

function addTransition() {
    imgBox.style.transition = "all .3s ease 0s";
    imgBox.style.webkitTransition = "all .3s ease 0s";
}

function removeTransition() {
    imgBox.style.transition = "none";
    imgBox.style.webkitTransition = "none";
}

function setTransfrom(t) {
    imgBox.style.transform = 'translateX(' + t + 'px)';
    imgBox.style.webkitTransform = 'translateX(' + t + 'px)';
}

// 开始动画部分
pointBox.children[0].className = "now";
for (var i = 0; i < ols.length; i++) {
    ols[i].index = i; // 获得当前第几个小 li 的索引号
    ols[i].onmouseover = function() {
        // 所有的都要清空
        for (var j = 0; j < ols.length; j++) {
            ols[j].className = "";
        }
        this.className = "now";
        setTransfrom(-indexx * width);
        square = indexx;
    }
}
timer = setInterval(function() {
```

```
    indexx++;
    addTransition();
    setTransfrom(-indexx * width);
    // 小方块
    square++;
    if (square > ols.length - 1) {
        square = 0;
    }
    // 先清除所有的
    for (var i = 0; i < ols.length; i++)
    {
        ols[i].className = "";
    }
    // 留下当前的
    ols[square].className = "now";
}, 3000);

imgBox.addEventListener('transitionEnd', function() {
    if (indexx >= 9) {
        indexx = 1;
    } else if (indexx <= 0) {
        indexx = 8;
    }
    removeTransition();
    setTransfrom(-indexx * width);
}, false);

imgBox.addEventListener('webkitTransitionEnd', function() {
    if (indexx >= 9) {
        indexx = 1;
    } else if (indexx <= 0) {
        indexx = 8;
    }
    removeTransition();
    setTransfrom(-indexx * width);
}, false);
```


```
/**
 * 触摸事件开始
 */
imgBox.addEventListener("touchstart", function(e) {
    console.log("开始");
    var event = e || window.event;
    //记录开始滑动的位置
    startX = event.touches[0].clientX;
}, false);

/**
 * 触摸滑动事件
 */
imgBox.addEventListener("touchmove", function(e) {
    console.log("move");
    var event = e || window.event;
    event.preventDefault();

    //清除定时器
    clearInterval(timer);
    //记录结束位置
    endX = event.touches[0].clientX;
    //记录移动的位置
    moveX = startX - endX;
    removeTransition();
    setTransfrom(-indexx * width - moveX);
}, false);

/**
 * 触摸结束事件
 */
imgBox.addEventListener("touchend", function() {
    console.log("end");
    //如果移动的位置大于三分之一，并且是移动过的
    if (Math.abs(moveX) > (1 / 3 * width) && endX != 0) {
```

```
        //向左
        if (moveX > 0) {
            indexx++;
        } else {
            indexx--;
        }
        //改变位置
        setTransfrom(-indexx * width);
    }
    //回到原来的位置
    addTransition();
    setTransfrom(-indexx * width);
    //初始化
    startX = 0;
    endX = 0;

    clearInterval(timer);
    timer = setInterval(function() {
        indexx++;
        square++;
        if (square > ols.length - 1) {
            square = 0;
        }
        // 先清除所有的
        for (var i = 0; i < ols.length; i++)
        {
            ols[i].className = "";
        }
        // 留下当前的
        ols[square].className = "now";
        addTransition();
        setTransfrom(-indexx * width);

    // 每 3 秒钟轮播图变化一次。
    }, 3000);
}, false);
```

```
};

module.exports = {
    bindEvent,
    scrollPic
}
```

（3）轮播图的视图层。

增加 src/components/HomeBanner.vue 文件，代码如下：

```
<template>
    <div class="home_ban">
        <div class="m_banner clearfix" id="my_banner">
            <ul class="banner_box">
                <!-- 更改这里就可以替换轮播图的图片了 -->
                <li><img
src="http://siweitech.b0.upaiyun.com/image/silulegou/2FgHsjCz7qfpSQr0.jpeg"
alt="" /></li>
                <li><img
src="http://siweitech.b0.upaiyun.com/image/silulegou/uJawxX6H3PBRcfMO.jpeg"
alt="" /></li>
            </ul>
            <ul class="point_box" >
                <li></li>
                <li></li>
            </ul>
        </div>
    </div>
</template>
```

这个 component 的内容非常简单，只是轮播图的 View。调用代码如下：

```
<HomeBannerView></HomeBannerView>
```

4. 增加物品分类

增加 src/components/HomeNav.vue：

```
<template>
    <div class="home_n">
```

```
        <nav class="m_nav">
            <ul>
                <li class="nav_item">
                    <a href="#" class="nav_item_link">
                        <img src="../assets/images/nav0.png" alt="">
                        <span>草原特色肉</span>
                    </a>
                </li>
                <li class="nav_item">
                    <a href="#" class="nav_item_link">
                        <img src="../assets/images/nav1.png" alt="">
                        <span>特色干果</span>
                    </a>
                </li>
                <li class="nav_item">
                    <a href="#" class="nav_item_link">
                        <img src="../assets/images/nav9.png" alt="">
                        <span>特色瓜子</span>
                    </a>
                </li>
                <li class="nav_item">
                    <a href="#" class="nav_item_link">
                        <img src="../assets/images/nav8.png" alt="">
                        <span>特色大米</span>
                    </a>
                </li>
            </ul>
        </nav>
    </div>
</template>
```

调用代码如下：

```
<HomeNavView></HomeNavView>
```

5. 效果图

默认页面效果如图 8-19 所示。

3 秒钟之后，轮播图发生了滚动，如图 8-20 所示。

图 8-19　默认页面效果

图 8-20　轮播图发生滚动

6. 总结

我们使用了轮播图组件，非常简单。步骤如下：

（1）复制对应组件的 js 文件到 src/lib 文件夹。
（2）复制对应组件的 vue 文件到 src/components 文件夹。
（3）在对应的 vue 文件中调用即可。

8.8 底部 Tab

页面的底部 Tab 是非常重要的部分，几乎所有的 H5 项目都会用到。

1. 在首页中引用底部 Tab

```
<template>
  <div class="background">
    <div class="home">
      <div class="m_layout">
        <!-- 轮播图-->
        <HomeBannerView></HomeBannerView>
      </div>
    </div>
    <!-- 这里就是底部 Tab -->
```

```
    <NavBottomView :is_shops_index="is_shops_index"></NavBottomView>
  </div>
</template>
<script>
    // 这里就是底部 Tab 对应的 vue 文件
    import NavBottomView from '../../components/NavBottom.vue';
</script>
```

2. 增加对应的 component 文件

增加/components/NavBottom.vue 文件，代码如下：

```
<template>
  <div class="footer">
      <footer class="fixBottomBox">
        <ul>
            <router-link tag="li" to="/" class="tabItem">
              <a href="javascript:;" class="tab-item-link" v-if="is_shops_index">
                <img src="../assets/footer01.png" alt="" class="tabbar-logo">
                <p class="tabbar-text" style="color: rgba(234, 49, 6, 0.66);">首页
</p>
              </a>
              <a href="javascript:;" class="tab-item-link" else>
                <img src="../assets/footer001.png" alt="" class="tabbar-logo">
                <p class="tabbar-text">首页</p>
              </a>
            </router-link>
            <router-link tag="li" to="/shops/category" class="tabItem">
               <a href="javascript:;" class="tab-item-link" v-if="is_category">
                  <img src="../assets/footer02.png" alt="" class="tabbar-logo">
                  <p class="tabbar-text" style="color: rgba(234, 49, 6, 0.66);">
分类</p>
               </a>
               <a href="javascript:;" class="tab-item-link" else>
                 <img src="../assets/footer002.png" alt="" class="tabbar-logo">
                 <p class="tabbar-text">分类</p>
               </a>
```

```
          </router-link>
          <router-link tag="li" to="/cart" class="tabItem">
              <a href="javascript:;" class="tab-item-link" v-if="is_cart">
                <img src="../assets/footer03.png" alt="" class="tabbar-logo">
                <p class="tabbar-text" style="color: rgba(234, 49, 6, 0.66);">购
物车</p>
              </a>
              <a href="javascript:;" class="tab-item-link" else>
                <img src="../assets/footer003.png" alt="" class="tabbar-logo">
                <p class="tabbar-text">购物车</p>
              </a>
          </router-link>
          <router-link tag="li" to="/mine" class="tabItem">
              <a href="javascript:;" class="tab-item-link" v-if="is_mine">
                <img src="../assets/footer04.png" alt="" class="tabbar-logo">
                <p class="tabbar-text" style="color: rgba(234, 49, 6, 0.66);">我
的</p>
              </a>
              <a href="javascript:;" class="tab-item-link" else>
                <img src="../assets/footer004.png" alt="" class="tabbar-logo">
                <p class="tabbar-text">我的</p>
              </a>
          </router-link>
      </ul>
    </footer>
  </div>
</template>

<script>
export default{
  data () {
    return {
    }
  },
  props: {
    is_shops_index: Boolean,
```

```
    is_category: Boolean,
    is_cart: Boolean,
    is_mine: Boolean,
  },
  mounted () {
  },
  computed: {
  },
  methods: {
  }
}
</script>
```

3. 效果图

效果如图 8-21 所示。

图 8-21　效果图

4. 总结

底部 Tab 很简单也很重要。在本节中使用了 component 实现，然后在其他页面重用。

8.9 商品列表页

因为商品列表出现在首页和列表页，所以可以直接把它做成组件。

下面以首页中引用为例进行讲解。

1. 在首页中添加代码

```
<template>
  <div class="background">
    <div class="home">
      <div class="m_layout">
        <div class="product_top">
          <div class="product_left">
            <div>商品列表</div>
          </div>
        </div>
        <span class="divider" style="height: 2px;"></span>
        <!-- 这里循环显示特产商品列表 -->

<SpecialMarket :id="good.id" :name="good.name" :description="good.description"
 :image_url="good.image_url" v-for="good in goods"></SpecialMarket>
      </div>
    </div>
    <NavBottomView :is_shops_index="is_shops_index"></NavBottomView>
  </div>
</template>
<script>
    // 在这里引入 特产 component
    import SpecialMarket from '../../components/SpecialMarket.vue';
</script>
```

核心代码如下：

```
<!-- 这里循环显示特产商品列表 -->
<SpecialMarket :id="good.id" :name="good.name" :description="good.description"
 :image_url="good.image_url" v-for="good in goods"></SpecialMarket>
```

上面代码使用了 v-for 和 componment 的组合。

2. 在 component 中添加文件

添加文件 src/components/SpecialMarket.vue。

```
<template>
  <div>
    <div @click="show_goods_details" class="fu_li_zhuan_qu" >
      <img :src="image_url" class="logo_image"/>
      <div class="content" >
        <div class="title">

        </div>
        <div class="logo_and_shop_name">
          <span v-html="description"></span>
        </div>
      </div>
    </div>
    <span class="divider" style="height: 2px;"></span>
  </div>
</template>
<script>
import { go } from '../libs/router'

    export default{
        data(){
            return {
            }
        },
        props: {
          id: Number,
          name: String,
          description: String,
          image_url: String,
        },
        mounted(){
        },
        methods:{
```

```
        show_goods_details () {
          go("/shops/goods_details?good_id=" + this.id, this.$router)
        },
      },
      components:{
      },
    }
</script>
```

可以看到，该段代码会接受一个数组，然后循环显示。点击某个商品就会跳转到该商品的详情页面。

3. 总结

这里非常简单，是基础知识。

8.10 商品详情页

当用户在商品列表页面中点击某个商品时，就会跳转到该页面。

1. 新增路由

向 src/router/index.js 中增加如下代码。

```
import GoodsDetails from '@/views/shops/goods_details'

Vue.use(Router)

export default new Router({
  routes: [
    {
      path: '/shops/goods_details',
      name: 'GoodsDetails',
      component: GoodsDetails
    },
  ]
})
```

2. 新增 vue 页面

向 src/views/shops/goods_details 中增加如下代码。

```
<template>
  <div class="background">
    <div class="goods_detail" style="height: 100%;">
      <header class="top_bar">
        <a onclick="window.history.go(-1)" class="icon_back"></a>
        <h3 class="cartname">商品详情</h3>
      </header>
      <div class="tast_list_bd" style="padding-top: 44px;">
        <main class="detail_box">

        <!-- 轮播图 -->
        <div class="home_ban">
          <div class="m_banner clearfix" id="my_banner">
            <ul class="banner_box" >
              <div v-for="image in good_images">
                <li><img :src="image" alt="" style="height: 300px"/></li>
              </div>
            </ul>
            <ul class="point_box" >
              <li></li>
            </ul>
          </div>
        </div>

        <section class="product_info clearfix">
          <div class="product_left">
            <p class="p_name"></p>
            <div class="product_pric">
              <span>¥</span>
              <span class="rel_price"></span>
              <span></span>

              <span style='color: grey;
```

```
            text-decoration: line-through;
            font-size: 18px;
            margin-left: 14px;'>
              原价：￥
            </span>
          </div>
          <!--
          <div class="product_right">
            降价通知
          </div>
          -->
        </div>
      </section>

      <span class="divider" style="height: 2px;"></span>
      <div id="choose_number" style= "height: 40px; background-color: #fff;">
        <label style="font-size: 18px; float: left; margin-left: 10.5px;
margin-top: 7.5px;">购买数量</label>
        <div style= "padding-top: 5px;">
          <img src="../../assets/add@2x.png" style="margin-right: 10px;
display: inline;float:right;width:30px;" class="plus" @click="plus"/>
          <input pattern="[0-9]*" v-model="buy_count" type="text"
name="counts" style="width:30px;display: inline;float:right;border: 0.5px
solid #e2e2e2;line-height:28px;text-align:center;"/>
          <img src="../../assets/minus@2x.png" style="display:
inline;float:right;width:30px;" class="minus" @click="minus"/>
        </div>
      </div>

      <section class="product_intro">
        <div class="pro_det" v-html="good.description" style='padding: 0
6.5px;'>
        </div>
      </section>
      </main>
    </div>
```

```
    <footer class="cart_d_footer">
      <div class="m">
        <ul class="m_box">
          <li class="m_item">
            <a @click="toCart" class="m_item_link">
              <em class="m_item_pic three"></em>
              <span class="m_item_name">购物车</span>
            </a>
          </li>
        </ul>
        <div class="btn_box clearfix" >
          <a @click="addToCart" class="buy_now">加入购物车</a>
          <a @click="zhifu" class="buybuy">立即购买</a>
        </div>
      </div>
    </footer>

    </div>
  </div>
</template>
<script>
import { go } from '../../libs/router'
//import { swiper, swiperSlide } from 'vue-awesome-swiper'
import {scrollPic} from '../../libs/index.js'

   export default{
       data(){
           return {
               good_images: [],
               good: "",
               buy_count: 1,
               good_id: this.$route.query.good_id,
           }
       },
       watch:{
```

```
    },
    mounted(){
      scrollPic();   //轮播图

      this.$http.get(this.$configs.api + 'goods/goods_details?good_id=' +
this.good_id).then((response)=>{
        console.info(this.good_id)
        console.info(response.body)
        this.good = response.body.good
        this.good_images = response.body.good_images

      },(error) => {
        console.error(error)
      });
    },
    methods:{
      addToCart () {
        alert("商品已经加入到了购物车")
        let goods = {
          id: this.good_id,
          title: this.good.name,
          quantity: this.buy_count,
          price: this.good.price,
          image: this.good_images[0]
        }
        this.$store.dispatch('addToCart', goods)
      },
      toCart () {
        go("/cart", this.$router)
      },
      plus () {
        this.buy_count = this.buy_count + 1
      },
      minus () {
        if(this.buy_count > 1) {
          this.buy_count = this.buy_count - 1
```

```
          }
        },
        zhifu () {
          go("/shops/dingdanzhifu?good_id=" + this.good_id + "&buy_count=" +
this.buy_count, this.$router)
        },
      },
      components: {
      },
            computed: {
            }
  }
</script>
```

在上面的代码中：

- 实现了加入购物车的方法；
- 实现了对于支付页面的跳转；
- 实现了从远程接口读取数据。

3. 添加商品到购物车

在 src/views/shops/goods_details.vue 文件，下面的代码是把某个商品添加到购物车中。

```
addToCart () {
  console.info('加入购物车')
  alert("商品已经加入到了购物车")
  let goods = {
    id: this.good_id,
    title: this.good.name,
    quantity: this.buy_count,
    price: this.good.price,
    image: this.good_images[0]
  }
  this.$store.dispatch('addToCart', goods)
},
```

我们需要在 src/vuex/actions.js 中添加如下代码。

```
export const addToCart = ({ commit }, product) => {
```

```
    commit(types.ADD_TO_CART, {
      id: parseInt(product.id),
      image: product.image,
      title: product.title,
      quantity: product.quantity,
      price: product.price
    })
}
```

4. 效果图

效果如图 8-22 所示。

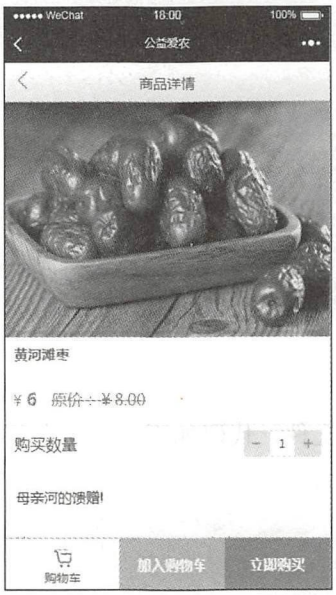


图 8-22　效果图

5. 总结

这个页面包含的知识点比较多，购物车使用了 Vuex 来保存数据。

进入支付页面，在后面会详述。本页面使用了后台提供的接口，会返回必要的数据。接口结构略。

8.11 购物车

购物车具备以下两个功能：

- 保存用户需要的数据；
- 清空商品。

所以使用 Vuex 来实现非常合适。

1. 添加路由

向文件 src/router/index.js 中增加如下代码。

```
import Cart from '@/components/Cart'

Vue.use(Router)

export default new Router({
  routes: [
    { path: '/cart',
      name: 'Cart',
      component: Cart
    },
  ]
})
```

2. 添加查看页面

新增 src/components/Car.vue 文件，代码如下：

```
<template>
  <div class="background">
    <div id="my_cart">
      <CartHeaderView></CartHeaderView>
      <CartMainView></CartMainView>
      <NavBottomView :is_cart="is_cart"></NavBottomView>
    </div>
  </div>
</template>

<script>

 import CartHeaderView from './CartHeader.vue';
 import CartMainView from './CartMain.vue';
```

```
import NavBottomView from './NavBottom.vue';

export default{
  data () {
    return {
      is_cart: true
    }
  },
  mounted(){
  },
  components:{
    CartHeaderView,
    CartMainView,
    NavBottomView
  }
}
</script>
```

3. 增加对应的组件

（1）新增购物车的头部文件 src/components/CartHeader.vue，代码如下：

```
<template>
   <div id="carttp">
      <header class="top_bar">
      <a onclick="window.history.go(-1)" class="icon_back"></a>
      <h3 class="cartname">购物车</h3>
   </header>
   </div>
</template>
```

（2）新增购物车的主体内容 src/components/CartMain.vue，代码如下：

```
<template>
   <main class="cart_box">
   <p v-show="!products.length"><i>请选择商品加入到购物车</i></p>
      <div class="cart_content clearfix" v-for="item in products"
style="position: relative;">
         <div class="cart_shop clearfix">
```

```
            <div class="cart_check_box">
                <div class="check_box" checked>
                </div>
            </div>
            <div class="shop_info clearfix">
                <span class="shop_name" style="font-size: 14px;">丝路乐购新疆商
城</span>
            </div>
        </div>

        <div @click="find_item_id(item)" class="cart_del clearfix"
style="display: inline-block; position: absolute; top: 10px; right: 10px;">
          <div class="del_top">
          </div>
          <div class="del_bottom">
          </div>
        </div>
        <div class="cart_item">
            <div class="cart_item_box">
                <div class="check_box">
                </div>
            </div>
            <div class="cart_detial_box clearfix">
                <a class="cart_product_link">
                    <img :src="item.image" alt="">
                </a>
                <div class="item_names">
                    <a>
                        <span></span>
                    </a>
                </div>
                <div class="cart_weight">
                    <span class="my_color" style="color: #979292;"></span>
                </div>
                <div class="cart_product_sell">
                    <span class="product_money">¥<strong
```

```
class="real_money"></strong></span>
                    <div class="cart_add clearfix">
                        <span @click="minus(item.id)" class="my_jian">-</span>
                        <input type="tel"
class="my_count" :value="item.quantity">
                        <span @click="add(item.id)" class="my_add">+</span>
                    </div>
                </div>
            </div>
        </div>

        <div class="pop" style="display: none">
          <div class="pop_box">
            <div class="del_info">
              确定要删除该商品吗?
            </div>
            <div class="del_cancel">
              取消
            </div>
            <div @click="deleteItem" class="del_ok">
              确定
            </div>
          </div>
        </div>

        <div class="cart_fo">
          <footer class="cart_footer">
            <div class="count_money_box">
              <div class="heji">
                <strong>合计:</strong>
                <strong style="color: #ff621a; font-size: 18px;">\{\{ total |
currency }}</strong>
              </div>
              <a :disabled="!products.length" @click="checkout(products)"
class="go_pay">
```

```
            <span style="color: #f5f5f5; font-weight: 600;">结算</span>
          </a>
        </div>
      </footer>
    </div>
  </main>
</template>
<script>
 import { mapGetters } from 'vuex'
 import { go } from '../libs/router'
 import {check,animatDelBox} from '../assets/js/cart.js'

   export default{
     data(){
       return{
         need_delete_item: {},
         cartDatas:[ ],
       }
     },
   mounted(){
     check();
     animatDelBox();
   },
   computed: {
     ...mapGetters({
       products: 'cartProducts',
       checkoutStatus: 'checkoutStatus'
     }),
     total () {
       return this.products.reduce((total, item) => {
         return total + item.price * item.quantity
       }, 0)
     }
   },
   methods: {
```

```
    // 跳转到支付页面
    checkout (products) {
      go("/shops/dingdanzhifu", this.$router)
    },

    // 对于商品的数量进行增加
    add (id) {
      this.$store.dispatch('changeItemNumber', {id, type: 'add'})
    },

    // 对于商品的数量进行减少
    minus (id) {
      this.$store.dispatch('changeItemNumber', {id, type: 'minus'})
    },

    // 删除某个商品
    deleteItem () {
      this.$store.dispatch('deleteItem', this.need_delete_item.id)
    },
    find_item_id (item) {
      this.need_delete_item = item
    }
  },
  }
</script>
```

（3）修改 Vuex 的函数，把下面代码添加到 src/vuex/actions.js 文件中。

```
export const deleteItem = ({ commit }, id) => {
  commit(types.DELETE_ITEM, {
    id: parseInt(id)
  })
}

export const changeItemNumber = ({ commit }, {id, type}) => {
  console.info(id)
  commit(types.CHANGE_ONE_QUANTITY, {
```

```
    id: parseInt(id),
    type
  })
}
```

上面的代码实现了购物车的若干功能，比如可以对商品数量进行增减；当商品数量改变时，商品总价也随之修改等。

4. 效果图

购物车页面如图 8-23 所示。

图 8-23　购物车页面

5. 总结

购物车的数据是通过 Vuex 保存的。

购物车中商品数量的加减是可以直接影响到商品总价的。这个功能使用 Vuex 实现非常适合，解决方案也非常优雅。如果使用传统的方式来做的话，代码就会非常臃肿，而且难以实现。

8.13 微信支付

微信支付最难的地方不在于技术，而是在于微信有一套自己的技术规范，建议读者查看官方文档。虽然有一些现成的工具集成（如 Ping++）了微信支付功能，但是往往这类产品入门门槛低，深入门槛高，后期交易量大了之后收费也很高昂，出了问题不好调试。另外，支付功能是核心技术，一定要亲自掌握，不要过于依赖第三方。

接下来的内容前提是微信的支付配置都已经做好了。

1. 添加支付页面的路由

为路由文件 src/router/index.js 增加如下代码。

```
import Pay from '@/components/pay'

Vue.use(Router)

export default new Router({
  routes: [
    {
      path: '/shops/pay',
      name: 'Pay',
      component: Pay
    },
  ]

})
```

2. 添加支付页面

新增 src/shops/pay.vue，内容如下：

```
<template>
  <div class="background">
    <header class="top_bar">
      <a onclick="window.history.go(-1)" class="icon_back"></a>
      <h3 class="cartname">订单支付</h3>
    </header>

    <div class="tast_list_bd" style="background-color: #F3F3F3; padding-top: 0;
```

```
padding-bottom: 80px;">
    <div class="goods_detail" style="">

      <main class="detail_box">
        <span class="divider"></span>

        <form style="margin-top: 45px;">
          <div class="column is-12">
            <label class="label">收货人</label>
            <p class="control has-icon has-icon-right">
            <input name="name" v-model="mobile_user_name" v-
validate="'required|required'" :class="{'input': true, 'is-danger':
errors.has('name') }" type="text" placeholder="例如: 张三"
autofocus="autofocus"/>
            <span v-show="errors.has('name')" class="help is-danger">收货人
不能为空</span>
            </p>
          </div>

          <div class="column is-12">
            <label class="label">收货地址</label>
            <p class="control has-icon has-icon-right">
            <input name="url" v-model="mobile_user_address" v-
validate="'required|required'" :class="{'input': true, 'is-danger':
errors.has('url') }" type="text" placeholder="例如: 北京市朝阳区大望路幸福里小区 8 栋
8 单元 118"/>
            <span v-show="errors.has('url')" class="help is-danger">收货地
址不能为空</span>
            </p>
          </div>

          <div class="column is-12">
            <label class="label">收货电话</label>
            <p class="control has-icon has-icon-right">
            <input name="phone" v-model="mobile_user_phone" v-
validate="'required|numeric'" :class="{'input': true, 'is-danger':
```

```
errors.has('phone') }" type="text" placeholder="例如: 18888888888"/>
                <span v-show="errors.has('phone')" class="help is-danger">电话
号码不能为空</span>
                </p>
              </div>
            </form>

        <span class="divider"></span>

        <section class="product_info clearfix" v-if="single_pay">
          <div>
            <div class="fu_li_zhuan_qu" >
              <img :src="good_images[0]" class="logo_image"/>
              <div class="content" >
                <div class="title">

                </div>
                <div class="logo_and_shop_name">
                  <div class="product_pric">
                    <span>¥</span>
                    <span class="rel_price"></span>
                    <span>   x </span>
                  </div>
                </div>
              </div>
            </div>
          </div>
        </section>

        <section class="product_info clearfix" v-else v-for="product in
cartProducts">
          <div>
            <div class="fu_li_zhuan_qu" >
              <img :src="product.image" class="logo_image"/>
              <div class="content" >
                <div class="title">
```

```
            </div>
            <div class="logo_and_shop_name">
              <div class="product_pric">
                <span>¥</span>
                <span class="rel_price"></span>
                <span>   x </span>
              </div>
            </div>
          </div>
        </div>
      </div>
    </section>

    <section>
      <span class="divider" style="height: 15px;"></span>
      <div class="extra_cost" style=" ">
        <span style="float: left; margin-left: 15px;"> 卖家留言:</span>
        <input v-model="guest_remarks" id="extra_charge" type="text"
name="cost" placeholder="选填: 对本次交易的说明" style="border: 0; background-
color: white;
          font-size: 15px; color: #48484b; outline: none; width:
60%;"></input>
      </div>
    </section>

    <section>
      <span class="divider"></span>
      <div class="extra_cost" style=" ">
        <span style="float: left; margin-left: 15px;"> 应付金额:</span>
        <div v-if="single_pay" class="rel_price" type="text" name="cost"
style="border: 0; background-color: white;
          font-size: 20px; color: #ff621a; font-weight: bold; outline: none;
text-align: right; padding-right: 20px;"> \{\{ total_cost | currency }}</div>
          <div v-else class="rel_price" type="text" name="cost" style="border:
0; background-color: white;
```

```
          font-size: 20px; color: #ff621a; font-weight: bold; outline: none;
text-align: right; padding-right: 20px;"> \{\{ total | currency }}</div>

        </div>
      </section>
      </main>

      <span class="divider"></span>

      <div style="height: 55px;  display: flex; width: 100%; padding: 0px 10px;
background-color: #fff;" @click="">
        <div style="flex: 1; display: flex;">
          <div style="margin-top: 10px;">
            <img src="../../assets/微信 icon@3x.png" style="width: 35px;"/>
          </div>
          <span style="margin-top: 8px; font-size: 18px; line-height:40px;
margin-left: 10px;">微信支付</span>
        </div>

        <div style=" padding: 14px 10px;" @click="user_wechat">
          <img src="../../assets/选中 3x.png" style="width: 28px;"/>
        </div>
      </div>
    </div>
  </div>

  <div class="shop_layout-scroll-absolute" style="">
    <div class="queding" @click="buy">
      立即支付
    </div>
  </div>

 </div>
</template>
<script>
  import { go } from '../../libs/router'
```

```
import { mapGetters } from 'vuex'
export default{
    data(){
        return {
            good_images: [],
            good: "",
            buy_count: this.$route.query.buy_count,
            good_id: this.$route.query.good_id,
            open_id: this.$store.state.userInfo.open_id,
            mobile_user_address: '',
            mobile_user_name: '',
            mobile_user_phone: '',
            guest_remarks: '',
            is_use_wechat: false,
        }
    },
    watch:{
    },
    mounted(){
      if (this.single_pay) {
          this.$http.get(this.$configs.api + 'goods/goods_details?good_id=' +
this.good_id).then((response)=>{
              console.info(this.good_id)
              console.info(response.body)
              this.good = response.body.good
              this.good_images = response.body.good_images
          },(error) => {
              console.error(error)
          });
      }
    },
    computed: {
        total () {
          return this.cartProducts.reduce((total, p) => {
            return (total + p.price * p.quantity)
          }, 0)
```

```
    },
    single_pay () {
      return this.good_id && this.buy_count
    },
    total_cost () {
      return this.good.price * this.buy_count
    },
    ...mapGetters({
      cartProducts: 'cartProducts',
      checkoutStatus: 'checkoutStatus'
    })
  },
  methods:{
    validateBeforeSubmit() {
      //拦截异步操作
      return new Promise((resolve, reject) => {
        this.$validator.validateAll().then(result => {
          console.info(result)
          if (result) {
            console.info("============表单验证成功===")
            resolve(true);
          } else {
            alert('请填写完整的收货信息!');
            resolve(false);
          }
        });
      })
    },
    plus () {
      this.buy_count = this.buy_count + 1
    },
    minus () {
      if(this.buy_count > 1) {
        this.buy_count = this.buy_count - 1
      }
    },
```

```
user_wechat () {
  if (this.is_use_wechat === false) {
    this.is_use_wechat = true
  } else {
    this.is_use_wechat = false
  }
},
buy (){
    let result = this.validateBeforeSubmit().then((resolve)=>{
        if (resolve) {
            console.info('true ==== ')
            let params
            if (this.single_pay) {
                params = {
                    good_id: this.good_id,
                    buy_count: this.buy_count,
                    total_cost: this.total_cost,
                    guest_remarks: this.guest_remarks,
                    mobile_user_address: this.mobile_user_address,
                    mobile_user_name: this.mobile_user_name,
                    mobile_user_phone: this.mobile_user_phone,
                    open_id: this.open_id
                }
            } else {
                console.info(this.total)
                params = {
                    goods: this.cartProducts,
                    total_cost: this.total,
                    guest_remarks: this.guest_remarks,
                    mobile_user_address: this.mobile_user_address,
                    mobile_user_name: this.mobile_user_name,
                    mobile_user_phone: this.mobile_user_phone,
                    open_id: this.open_id
                }
            }
            this.$http.post(this.$configs.api + 'goods/buy', params
```

```
                    ).then((response) => {
                        let order_number =  response.body.order_number
                        this.purchase(order_number)
                    }, (error) => {
                        console.error(error)
                    });
                } else {
                    console.info('== 请填写完整的收货信息')
                }
            });
        },
        purchase (order_number) {
          //调起微信支付界面
          if (typeof WeixinJSBridge == "undefined"){
            if( document.addEventListener ){
              document.addEventListener('WeixinJSBridgeReady',
this.onBridgeReady, false);
            }else if (document.attachEvent){
              document.attachEvent('WeixinJSBridgeReady',  this.onBridgeReady);
              document.attachEvent('onWeixinJSBridgeReady',
this.onBridgeReady);
            }
          }else{
            this.onBridgeReady(order_number);
          }
        },
        onBridgeReady (order_number) {
          let that = this
          let total_cost
          if (this.single_pay) {
            total_cost = this.total_cost
          } else {
            total_cost = this.total
          }
          this.$http.post(this.$configs.api + 'payments/user_pay',
          {
```

```
                open_id: this.$store.state.userInfo.open_id,
                total_cost: total_cost,
                order_number: order_number
              }).then((response) => {
                WeixinJSBridge.invoke(
                    'getBrandWCPayRequest', {
                        "appId": response.data.appId,
                        "timeStamp": response.data.timeStamp,
                        "nonceStr": response.data.nonceStr,
                        "package": response.data.package,
                        "signType": response.data.signType,
                        "paySign":  response.data.paySign
                    },
                    function(res){
                        // 下面代码仅用于调试
                        // alert("res.err_msg: " + res.err_msg + ", err_desc: " +
res.err_desc)
                        if(res.err_msg == "get_brand_wcpay_request:ok" ) {
                          // 使用以上方式判断前端返回,微信团队郑重提示: res.err_msg 将在用户
支付成功后返回    ok，但并不保证它绝对可靠。
                          that.$router.push({ path: '/shops/paysuccess?order_id=' +
order_number });
                        } else {
                          // 显示取消支付或失败
                          that.$router.push({ path: '/shops/payfail?order_id=' +
order_number });
                        }
                    }
                );
              }, (error) => {
                console.error(error)
              });
            }
          },
    }
</script>
```

核心代码如下：

```
onBridgeReady (order_number) {
  //....
  this.$http.post(this.$configs.api + 'payments/user_pay',
  {
    open_id: this.$store.state.userInfo.open_id,
    total_cost: total_cost,
    order_number: order_number
  }).then((response) => {
    WeixinJSBridge.invoke(
        'getBrandWCPayRequest', {
            "appId": response.data.appId,
            "timeStamp": response.data.timeStamp,
            //....
        },
        function(res){
            //...
        }
    );
  }, (error) => {
    console.error(error)
  });
}
```

上面的代码用于页面一准备好（即 WeixinJSBridge 准备好了）的时候，当前页面就要调用的。

```
purchase (order_number) {
  //跳到微信支付页面
  if (typeof WeixinJSBridge == "undefined"){
    if( document.addEventListener ){
      document.addEventListener('WeixinJSBridgeReady', this.onBridgeReady,
false);
    }else if (document.attachEvent){
      document.attachEvent('WeixinJSBridgeReady', this.onBridgeReady);
      document.attachEvent('onWeixinJSBridgeReady', this.onBridgeReady);
    }
  }else{
```

```
    this.onBridgeReady(order_number);
  }
},
```

上面的代码用于打开微信支付页面。其中 WeixinJSBridge 是微信浏览器自带的变量，不必声名直接调用即可。

4. 效果图

微信的支付页面打开（图略）。

5. 总结

- 微信支付的细节处理都交给了后端处理。只要前端把参数准备好，直接访问后端链接 http://shopweb.siwei.me/api/payments/user_pay 即可。
- 新手对于变量 WeixinJSBridge 很难掌握，需要多查看文档。另外，微信支付对于后端开发难度更高，建议后端开发人员多查多试。
- 在微信的后台要配置不同的支付目录。安卓和 iOS 的配置是不一样的。建议大家百度一下。
- 微信的支付场景对应的支付方式和实现方式是不一样的。本例是“微信的公众号内支付”。
- 微信官方文档提供的例子仅有基于经典的 Web 页面（非 SPA）的情况，目前还没有看到 SPA 的例子。建议大家遇到问题多上网搜索。

由于篇幅限制，微信相关的内容不再赘述。

8.14 回顾

本章我们把一个公益扶农的项目从零到一搭建了起来。实际上这是笔者公司参与的一个真实项目，我们使用 Vue.js 开发之后，很快就交付给了甲方使用。

使用 Vue.js 来做开发，有以下好处：

- 开发效率更高。
- 页面解耦更加彻底，整体结构清晰，文件组织合理。
- 可以很方便地引用现成的组件。
- 就算遇到难题，也比使用其他的框架简单不少。
- 自带的双向绑定，极大地节省了开发时间。
- 前/后端分离的非常彻底，特别适合做微信端、H5 端的开发。

注意：为了节约篇幅，本章的代码中都省去了<style>的内容。可以在以下网址找到完整的源代码：https://github.com/sg552/happy_book_vuejs。

Vue.js快速入门

与传统的Web应用不同，单页应用在近几年发展迅猛。国外的典型产品是Gmail，为传统的Web页面注入了意想不到的活力，国内则是由于微信的迅猛发展，单页应用在手机端表现性能十分优异，使用越来越流行。

目前来看，Vue.js作为极其耀眼的项目之一，学习起来快速直接，受到国内开发人员的推崇，许多知名应用也纷纷使用本框架。可以认为，Vue.js框架直接冲击了老牌的Angular和React，是未来的主流单页应用框架。

清华社官方微信号

扫 我 有 惊 喜

ISBN 978-7-302-51754-2

9 787302 517542 >

定价：59.00元